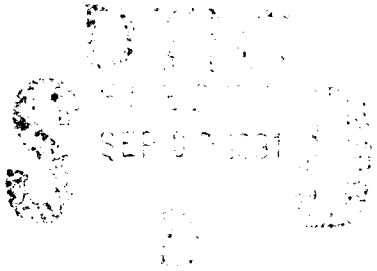
Semi-Annual Report

DEVELOPMENT OF PARALLEL ARCHITECTURES FOR SENSOR ARRAY PROCESSING ALGORITHMS



Submitted to:

Department of the Navy
Office of the Chief of the Naval Research
Arlington, VA 22217-5000

Submitted by:

M. M. Jamali Principal Investigator

S. C. Kwatra Co-Investigator

AbdelHamid Djoudi Research Associate

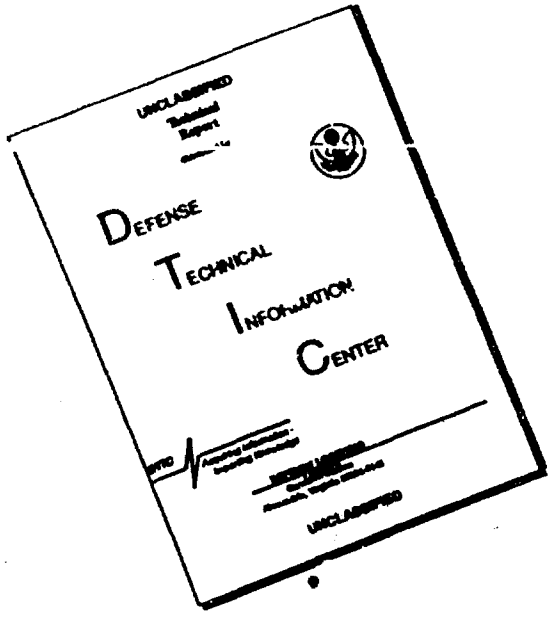
Rajesh Sheelvant Graduate Research Assistant

Manoj Rao Graduate Research Assistant

Department of Electrical Engineering
College of Engineering
The University of Toledo
Toledo, Ohio 43606

August 1991

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

ABSTRACT

The high resolution direction-of-arrival (DOA) estimation has been an important area of research for a number of years. Many researchers have developed a variety of algorithms to estimate the direction of arrival. Another important aspect of the DOA estimation area is the development of high speed hardware capable of computing the DOA in real time. In this research we have first focussed on the development of parallel architecture for multiple signal classification (MUSIC) and estimation os signal parameters by rotational invariance technique (ESPRIT) algorithms for the narrow band sources. These algorithms are substituted with computationaly efficient modules and converted to pipelined and parallel algorithms. For example one important computation of eigendecomposition of the covariance matrix has been performed using Householders transformations and QR method. MUSIC/ESPRIT algorithms are also shown in detailed pipelined flowchart. Systolic architectures are developed for both MUSIC/ESPRIT algorithms. Two other approaches of using cordic processors and single instruction multiple data (SIMD) machines for the computation of MUSIC/ESPRIT algorithms are also being studied.

The second part of this research is to perform DOA estimation for the wideband sources. Current literature is being studied and at the present time three algorithms are under investigation. For one of the three algorithms we have proposed a computationally simple algorithm and its flowchart is also shown.

Chapter I presents theoretical and mathematical aspects of MUSIC/ ESPRIT algorithms. These algorithms are modified and parallelized and described in Chapter II. Hardware implementations of tnese algorithms are given in Chapter III.

Cordic Processor approach is shown in Chapter IV. Wideband case is presented in chapter V.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION TO ARRAY SIGNAL PROCESSING

1.1: INTRODUCTION

The high resolution direction-of-arrival (DOA) estimation is important in many sensor systems. It is based on the processing of the received signal and extracting the desired parameters of the DOA of plane waves. Many approaches have been used for the purpose of implementing the function required for the DOA estimation including the so called maximum likelihood (ML) and the maximum entropy (ME) methods [1-3]. Although they are widely used, they have met with only moderate success. The ML method yields to a set of highly non linear equations, while the ME introduces bias and sensibility parameters estimates due to use of an incorrect mode (e.g. AR rather than ARMA). The Multiple Signal Classification (MUSIC) and the Estimation of Signal Parameters by Rotational Invariance techniques (ESPRIT) algorithms are two novel approaches used recently to provide asymptotically unbiased and efficient estimates of the DOA [4][5]. They are believed to be the most promising and leading candidates for further study and hardware implementation for real time applications. They estimate the so called signal subspace from the array measurements. The parameters of interest (i.e. determining of the DOA) are then estimated from the intersection between the array manifold and the estimated subspace.

1.2: DATA MODEL

For the purpose of understanding the advantages of using a sensor array in

DOA estimation, it is necessary to explore the nature of signals and noise the array is desired to receive. It is well known that in active sensing situations, the scattered data fluctuates randomly about a true value representing a noise free signal. This is due to noise effects and errors in a sensor array system. These fluctuations can be both additive and multiplicative. The additive fluctuations are due to thermal noise, shot noise, atmospheric noise, and other kinds of noise which are independent of the desired signal. The multiplicative fluctuations are due to measured errors in estimating the signal amplitudes, gain variation, etc. A noise model that represents all these noise effects is, in general, difficult to obtain, especially when some of the noise sources are dominant. Usually, based on the noise models, additive and/or multiplicative, the calculated probability of error, as a function of the noise power, is practically similar in each case. This indicates that the noise power rather than its specific characteristics, has more impact on the sensor array performance. Moreover, one is usually concerned with the effects of the additive noise on the output of a sensor array system. For this reason, an additive noise would be appropriate to choose for the evaluation of the performance of a system. This noise represents the totality of small independent sources, and by virtue of the central limit theorem one can model the resulting noise as Gaussian and (usually) stationary process. Also, to make the problem analytically tractable, we focus on narrow band signals where it is assumed that the power of all emitter signals is concentrated in the same narrow frequency band. In this context, two more assumptions that are of interest are invoked. First, we assume that the sources are in the far field of the array, consequently the radiation impinging on the array is in the form of plane waves, and secondly, the transmission medium is assumed to be isotropic so that the radiation propagates in straight line. Based on these assumptions, the output of any array element can be represented by a time advanced version or time delayed version of the received signal at a reference

element as shown in Figure 1.1.

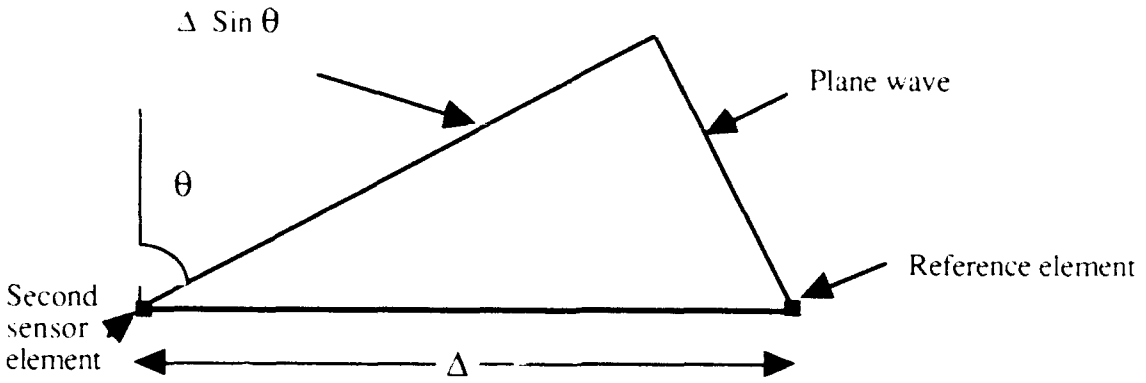


Figure 1.1: A two sensor array

Since the narrow-band signals are assumed to have the same known frequency ω, the received signals at the reference sensor and the second sensor are respectively given by

$$s(t) = u(t) \exp[\, j(\omega_o t + v(t))] \tag{1.1}$$

$$s(t-\tau) = u(t-\tau) \exp[\, j(\omega_o (t - \tau) + v(t- \tau))] \tag{1.2}$$

where u(t), and v(t) are the amplitude and phase of s(t) respectively. The signal s(t- τ) at the second sensor is delayed by the time required for the plane wave to propagate through Δ sin θ, and if c represents the velocity of propagation, then this time delay τ is given by

$$\tau = \frac{\Delta \sin \theta}{c} \tag{1.3}$$

The narrow band assumption implies that u(t) and v(t) are slowly varying functions, thus:

functions, thus:

$$u(t) = u(t-\tau) \tag{1.4}$$

$$v(t) = v(t-\tau) \tag{1.5}$$

for all possible propagation delays. Thus the effect of a time delay on the received signal is simply a phase shift:

$$s(t-\tau)= s(t)\exp(-j\omega_o\tau) \tag{1.6}$$

Now consider an array consisting of m sensors and receiving signals from d sources located at directions $\theta_1\ \theta_2, \ldots\ \theta_d$ with respect to the line of array, as shown in Figure 1.2.

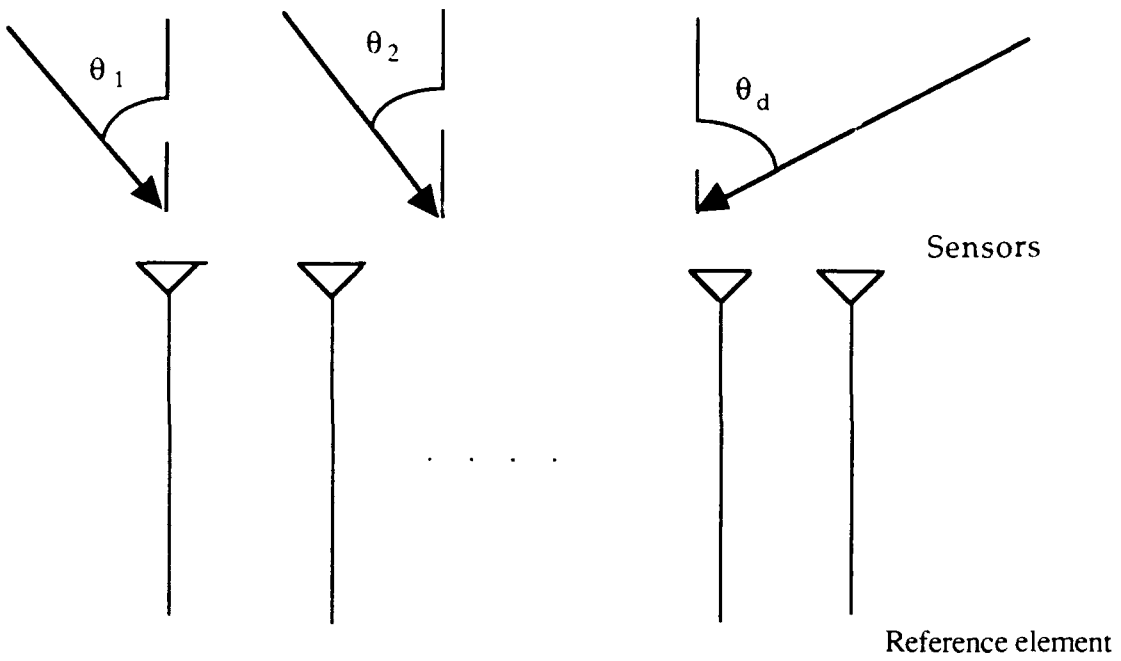


Figure 1.2: Typical array scene.

It is assumed that none of the signals are coherent (i.e. $|\rho_{ij}| \neq 1$). Using superposition of signal contribution, the received signal at the k^{th} sensor can be written as

$$x_k(t) = \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t-\tau_k(\theta_i)) + n_k(t)$$

$$= \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t) \exp(-j\omega_o \tau_k(\theta_i)) + n_k(t) \qquad (1.7)$$

where $\tau_k(\theta_i)$ is the propagation delay between a reference point and the k^{th} sensor for the i^{th} wavefront impinging on the array from direction θ_i, $a_k(\theta_i)$ is the corresponding sensor element complex response (gain and phase) at frequency ω_o and $n_k(t)$ stands for the additive noise at the k^{th} sensor. If we let

$$\mathbf{a}(\theta_i) = \{ a_1(\theta_i)\exp(j\omega_o\tau_1(\theta_i)) \ldots a_m(\theta_i)\exp(jw_o\tau_m(\theta_i))\}^H \quad (1.8)$$

Where denotes complex conjugate transpose
and $n(t) = \{ n_1(t), n_2(t) \ldots\ldots\ldots\ldots n_m(t)\}^T$

the data model representing the outputs of m sensors becomes

$$\mathbf{x}(t)=\sum_{i=1}^{d}\mathbf{a}(\theta_i)s_i(t) + \mathbf{n}(t) \quad (1.9)$$

Now by setting

$$\mathbf{A}(\theta) =(\mathbf{a}(\theta_1), \mathbf{a}(\theta_2), \ldots , \mathbf{a}(\theta_d)) \quad (1.10)$$

and

$$\mathbf{s}(t) = (s_1(t), s_2(t), \ldots , s_d(t))^T \quad (1.11)$$

x(t) can be rewritten as

$$\mathbf{x}(t) = \mathbf{A}(\theta)\,\mathbf{s}(t) + \mathbf{n}(t) \quad (1.12)$$

where

$$\mathbf{x}(t),\ \mathbf{n}(t) \in C^m,\ \mathbf{s}(t) \in C^d \quad \text{and} \quad \mathbf{A}(\theta) \in C^{m\times d}$$

$\mathbf{A}(\theta)$ is called the direction matrix. The columns of $\mathbf{A}(\theta)$ are elements of a set, termed the array manifold, composed of all array response vectors obtained as ranges over the entire space. If we assume that signals and noise are stationary, zero mean, uncorrelated random processes and further the noises in different sensors are uncorrelated, the spatial correlation matrix of the observed signal vector x(t) is defined by:

$$\mathbf{R}_{xx} = E\,(\mathbf{x}(t)\,\mathbf{x}^H(t)) \quad (1.13)$$

where E is the expectation operator.

The substitution of Equation (1.12) into (1.13) gives

$$\mathbf{R}_{xx} = E\left(\mathbf{A}(\theta)\, \mathbf{s}(t)\, \mathbf{s}(t)^{H} (\mathbf{A}(\theta))^{H} + \sigma^2 . I \right)$$

$$= \mathbf{A}(\theta)\ \mathbf{R}_{ss}\ \mathbf{A}(\theta)^{H} + \sigma^2 \mathbf{I} \qquad (1.14)$$

where

$$\mathbf{R}_{ss} = E(\mathbf{s}(t)\mathbf{s}(t)^{H}) \qquad (1.15)$$

and $\sigma^2 . \mathbf{I}$ is the spatial correlation matrix of the noise vector n(t), σ^2 denotes the variance of the elemental noise $n_i(t)$. i = 1, ... n

1.3: MULTIPLE SIGNAL CLASSIFICATION (MUSIC) ALGORITHM

Consider first the noise free case where

$$x(t) = \sum_{i=1}^{d} \mathbf{a}(\theta_i)\, s_i(t) \qquad (1.16)$$

This means that x(t) is a linear combination of the d steering column vectors of $\mathbf{A}(\theta)$ and is therefore constrained to the d-dimensional subspace of $\mathbf{C}_{m,}$, termed the signal subspace, that is spanned by the d columns vectors of $\mathbf{A}(\theta)$. In this case the signal subspace intersects the array manifold at the d steering vectors $\mathbf{a}(\theta_i)$ as shown in Figure 1.3.

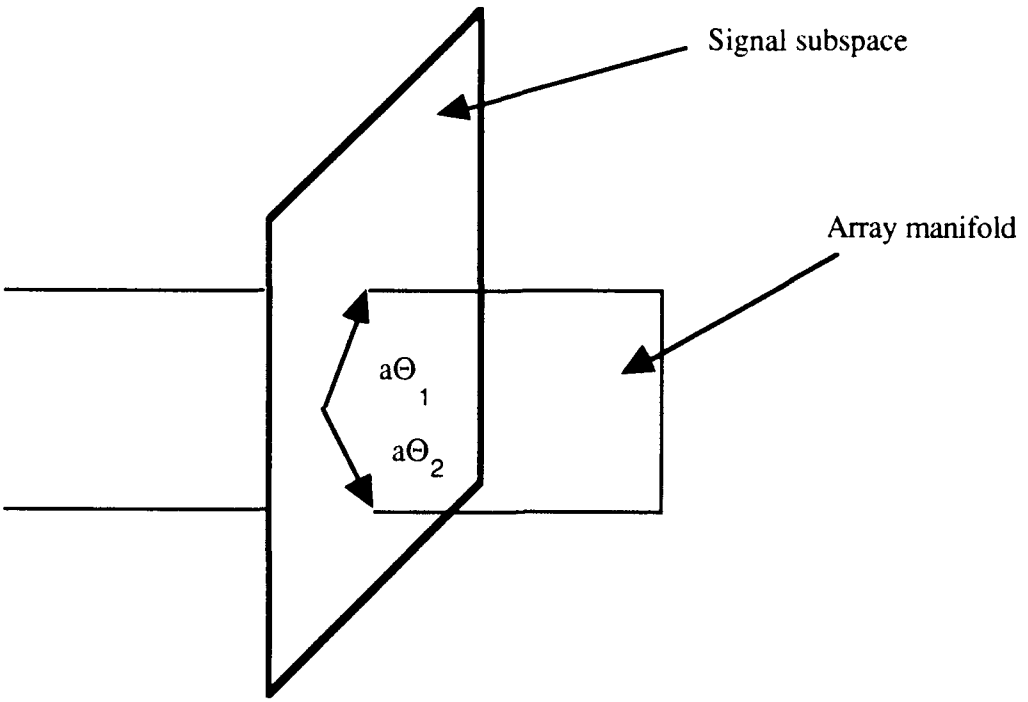


Figure 1.3: Signal subspace and array manifold for a two-source example.

However, when the data is corrupted by noise, the signal subspace has to be estimated and consequently it is expected that the signal subspace will not intersect the array manifold, so the steering vectors closest to the signal subspace will be chosen instead [6]. In the following, it is shown that one set of d independent vectors that span the signal subspace is given by the d eigenvectors corresponding to the d largest eigenvalues of the data covariance matrix. The data covariance matrix is assumed to be positive definite and Hermetian and consequently its eigendecomposition is given by

$$\mathbf{R}_{xx} = \mathbf{E}\,\Lambda\,\mathbf{E}^{H} \qquad (\mathbf{E}\,\mathbf{E}^{H} = \mathbf{I})$$

$$\Rightarrow \quad \mathbf{R}_{xx}\,\mathbf{E} = \mathbf{E}\,\Lambda$$

$$\Rightarrow \quad (\mathbf{A}(\theta)\mathbf{R}_{ss}\mathbf{A}(\theta)^{H} + \sigma^2.\mathbf{I})\,\mathbf{E} = \mathbf{E}\,\Lambda$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{E} = \mathbf{E}\,\Lambda - \sigma^2.\mathbf{E}$$

$$\Rightarrow \quad \mathbf{E}^{H}\,\mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{E} = \mathbf{E}^{H}\,\mathbf{E}\,\Lambda - \sigma^2.\mathbf{E}^{H}\mathbf{E}$$

$$= \Lambda - \sigma^2.\mathbf{I}$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H} = \mathbf{E}(\Lambda - \sigma^2.\mathbf{I})\,\mathbf{E}^{H} \qquad (1.17)$$

Thus the eigenvalues of $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}$ are the d largest eigenvalues of $\mathbf{R}_{xx}$ augmented by σ^2. Also the (m-d) smallest eigenvalues are all equal to σ^2. Now if $(\lambda_i, \mathbf{e}_i)$ is an eigenpair of $\mathbf{R}_{xx}$, then

$$\mathbf{R}_{xx}\,\mathbf{e}_i = \lambda_i\,\mathbf{e}_i \qquad (1.18)$$

and for any i > d,

$$(\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H} + \sigma^2.\mathbf{I})\,\mathbf{e}_i = \sigma^2\mathbf{e}_i$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{e}_i = 0 \qquad (1.19)$$

Now from the fact that $\mathbf{A}(\theta)$ and $\mathbf{R}_{ss}$ must have at least one nonsingular submatrix of order d and without loss of generality, suppose that this submatrix consists of the first d rows of $\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}$. Partition $\mathbf{A}_1(\theta)\mathbf{R}_{ss}$ as:

$$\mathbf{A}(\theta)\,\mathbf{R}_{ss} = (\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}\,,\,\mathbf{A}_2(\theta)\,\mathbf{R}_{ss})^{T} \qquad (1.20)$$

The substitution of (1.20) into (1.19) yields

$$\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{e}_i = 0 \qquad (1.21)$$

and

$$\mathbf{A}_2(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^H\,\mathbf{e}_i = 0 \qquad (1.22)$$

For the equation (1.21) to be satisfied,

$$\mathbf{A}(\theta)^H\,\mathbf{e}_i = 0 \qquad i > d \qquad (1.23)$$

Thus $\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d$ span the same subspace spanned by the column vectors of $\mathbf{A}(\theta)$. In most situations, the covariance matrices are not known exactly but need to be estimated. Therefore, one can expect that there is no intersection between the array manifold and the signal subspace. However, elements of the array manifold closest to the signal subspace should be considered as potential solution. After determining the number of sources [7], Smith [5] proposed the following function as one possible measure of closeness of an element of the array manifold to the signal subspace

$$P_m(\theta) = \frac{1}{\mathbf{a}^H(\theta)\,\mathbf{E}_n\mathbf{E}^H{}_n\,\mathbf{a}(\theta)} \qquad (1.24)$$

where $\mathbf{E}_n = [\,\mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots, \mathbf{e}_m\,]$

The dominant d peaks over $\theta \in [-\pi, \pi]$ are the desired estimates of the directions of arrival.

For the particular case where the array consists of m sensors uniformly spaced, and if the reference point is taken at the first element of the array, $P_m(\theta)$ is obtained by first calculating the DFT of the vectors spanning the null space of $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^H$ or

$$\mathbf{E}_n = [\,\mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots, \mathbf{e}_m] \qquad (1.25)$$

If Δ is the distance separating two sensors of the array, an element of the array manifold is given by

$$\mathbf{a}(\theta) = (1 , \exp (j2 \pi\Delta \sin\theta / \lambda), \ldots , \exp (j2 \pi(m-1)\Delta \sin\theta / \lambda))^T \qquad (1.26)$$

and the DFT of the vector $\mathbf{e}_i$, $i > d$ is given by

$$F_i = \mathbf{a}^*(\theta)\, \mathbf{e}_i = \sum_{i=1}^{m} \mathbf{e}_{ki} \exp(-j2 \pi(k-1)\Delta \sin\theta / \lambda) \qquad (1.27)$$

thus

$$P_m(\theta) = \frac{1}{\sum_{k=1}^{m} |F_i|^2} \qquad (1.28)$$

Summary of the MUSIC algorithm

1) Estimate the data covariance matrix **R.**

2) Perform the eigendecomposition of **R**.

3) Estimate the number of sources.

4) Evaluate $P_m(\theta)$.

5) Find the d largest peaks of $P_m(\theta)$ to obtain estimates of the parameters

Although MUSIC is a high resolution algorithm, it has several drawbacks including the fact that complete knowledge of the array manifold is required, and that is computationaly very expensive as it requires a lot of computations to find the intersection between the array manifold and the signal subspace. In the next section, another algorithm known as ESPRIT will be discussed. Even though it is similar to

the MUSIC and exploits the underlying data model but it eliminates the requirement of a time-consuming parameter search.

1.4: ESTIMATION OF SIGNAL PARAMETERS VIA ROTATIONAL INVARIANCE (ESPRIT)

Consider a planar array composed of m pairs of pair identical sensors (doublets) as shown in the Figure 1.4. The displacement between two sensors in each doublet is constant, but the sensor characteristics are unknown [5].

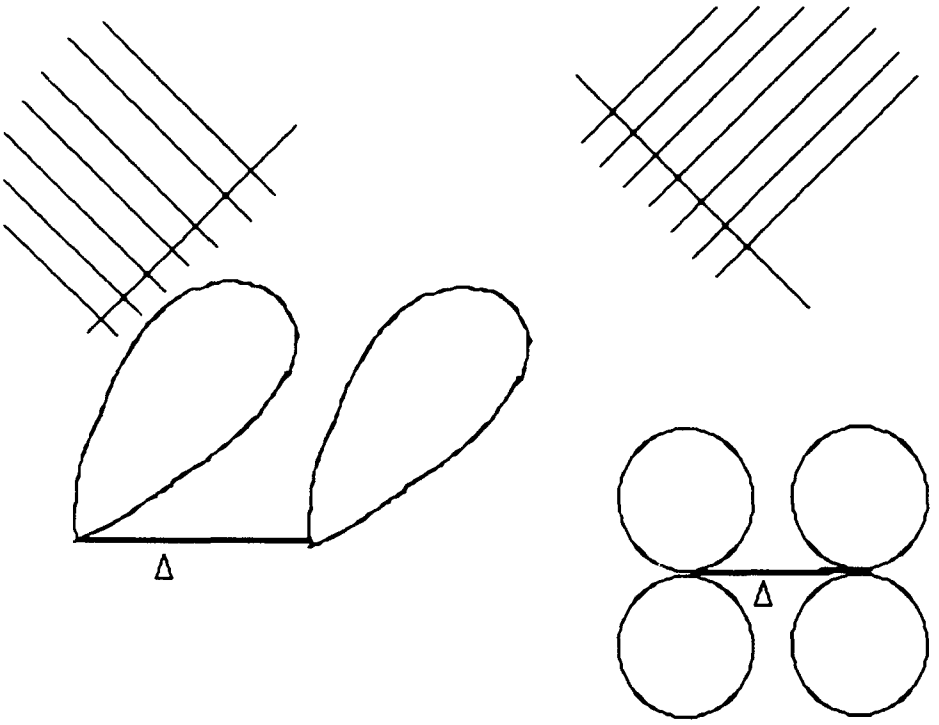


Figure 1.4: Sensor array for ESPRIT.

The signal received at the i^{th} doublet can be expressed as

$$x_k(t) = \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t) + n_{xk}(t)$$

$$y_k(t) = \sum_{i=1}^{d} a_k(\theta_i) \exp(j\omega_o \Delta \sin \theta_i / c)\, s_i(t) + n_{yk}(t) \qquad (1.29)$$

where θ_i is the direction of arrival of the i^{th} source relative to the direction of translational displacement vector. Employing vector notation as in the case of MUSIC, the data vector can be expressed as:

$$\mathbf{x}(t) = \mathbf{A}(\theta)\, \mathbf{s}(t) + \mathbf{n}_x(t)$$

$$\mathbf{y}(t) = \mathbf{A}(\theta)\Phi\, \mathbf{s}(t) + \mathbf{n}_y(t) \qquad (1.30)$$

where

$$\Phi = \mathrm{diag}\left(\exp\left(\frac{j\omega_o \Delta \sin\theta_1}{c}\right), \ldots, \exp\left(\frac{j\omega_o \Delta \sin\theta_d}{c}\right)\right)$$

Now, consider the matrices

$$\mathbf{C}_{xx} = \mathbf{R}_{xx} - \sigma^2 \mathbf{I}$$

$$= \mathbf{A}(\theta)\, \mathbf{R}_{ss}\, \mathbf{A}^*(\theta) \qquad (1.31)$$

and

$$\mathbf{R}_{xy} = \mathbf{A}(\theta)\, \mathbf{R}_{ss}\, \Phi^* \mathbf{A}^*(\theta) \qquad (1.32)$$

In the computation of $\mathbf{R}_{xx}$ the noise in different sensors is assumed to be uncorrelated ($\mathbf{E}[n_x(t)\, n_y(t)] = 0$).

The eigenvalues of the matrix pencil ($\mathbf{C}_{xx}$, $\mathbf{R}_{xy}$) are obtained by solving

$$\left| \mathbf{C}_{xx} - \gamma \mathbf{R}_{xy} \right| = 0 \qquad (1.33a)$$

or

$$\left| \mathbf{A}(\theta)\, \mathbf{R}_{ss} (\mathbf{I} - \gamma \Phi^{*}) \mathbf{A}^{*}(\theta) \right| = 0 \qquad (1.33b)$$

Now from the fact that $\mathbf{A}(\theta)$ and $\mathbf{R}_{ss}$ are full rank matrices, Equation (1.33b) reduces to

$$\left| \mathbf{I} - \gamma \Phi^{*} \right| = 0 \qquad (1.34)$$

and the desired singular values are

$$\gamma_k = \exp\left(\frac{j\omega_o \Delta \sin\theta_k}{c}\right) \quad k=1,\ldots,d \qquad (1.35)$$

Thus the direction of arrival can be obtained without involving a search technique as in the MUSIC case, and in that respect computation and storage costs are reduced considerably. Also it can be concluded can conclude that the generalized eigenvalue matrix associated with the matrix pencil $(\mathbf{C}_{xx'}, \mathbf{R}_{xy})$ is given by:

$$\mathbf{A} = \begin{vmatrix} \Phi & 0 \\ 0 & 0 \end{vmatrix} \qquad (1.36)$$

However, due to error in estimating $\mathbf{R}_{xx}$ and $\mathbf{R}_{xy}$ from a finite data sample as well as round-off errors introduced during the squaring of the data, the relation between $\mathbf{A}$ and Φ given above is not exactly satisfied, which make this method suboptimal.

The following procedure is proposed to estimate the generalized eigenvalues [7]

1) Find the data covariance matrix of the complete 2m sensors, denoted by $\mathbf{R}_{zz}$.

2) Estimate the number of sources d.

3) Estimate the noise variance (average of the 2m - d noise eigenvalues).

4) Compute $\mathbf{R}_{zz}$- $\sigma^2 I$, $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}^*(\theta)$ and $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\Phi^*\mathbf{A}^*(\theta)$ are then the top left and top right blocks.

5) Calculate the generalized eigenvalues of the matrix pencil ($\mathbf{C}_{xx}$, $\mathbf{R}_{xy}$) and choose the d ones that lie close to the unit circle.

1.5: TOTAL LEAST SQUARE (TLS) ESPRIT

The last method is based on having a very good estimate of the noise variance, a condition difficult to satisfy in most real cases. This may yield overall inferior results. To circumvent this difficulty to some extent, the total-least-square (TLS ESPRIT) scheme is used instead.

Let

$$z(t)=\begin{vmatrix} x(t) \\ y(t) \end{vmatrix} = \overline{\mathbf{A}}\, s(t) + \mathbf{n}_z(t) \tag{1.37}$$

where

$$\overline{\mathbf{A}} = \begin{vmatrix} \mathbf{A} \\ \mathbf{A}\Phi \end{vmatrix}, \quad \mathbf{n}_z(t) = \begin{vmatrix} \mathbf{n}_x(t) \\ \mathbf{n}_y(t) \end{vmatrix} \tag{1.38}$$

and let $\mathbf{E}_s = [\,\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d\,]$ be the $(2m \times d)$ matrix composed of the eigenvectors corresponding to the d largest eigenvalues of ($\mathbf{R}_{zz}$, I). Since the columns of $\mathbf{E}_s$ and $\overline{\mathbf{A}}$ span the same subspace, then there must exist a non-signular $(d \times d)$ $\mathbf{T}$ matrix such that

$$\mathbf{E}_s = \overline{\mathbf{A}}\ \mathbf{T} \tag{1.39}$$

Now define two m × d matrices $\mathbf{E}_x$ and $\mathbf{E}_y$ by partitioning $\mathbf{E}_s$ as

$$\mathbf{E}_s = \begin{vmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{vmatrix} = \begin{vmatrix} \mathbf{AT} \\ \mathbf{A\Phi T} \end{vmatrix} \tag{1.40}$$

Since $\mathbf{E}_x$ and $\mathbf{E}_y$ share a common space (i.e. the columns of both $\mathbf{E}_x$ and $\mathbf{E}_y$ are a linear combination of the columns of A), then the rank of $\mathbf{E}_{xy} = [\mathbf{E}_x \mid \mathbf{E}_y]$ is d which implies that there exist a unique 2d × d matrix F of rank d such that

$$0 = [\mathbf{E}_x \mid \mathbf{E}_y]\ \mathbf{F} = \mathbf{E}_x\mathbf{F}_x + \mathbf{E}_y\mathbf{F}_y$$

$$= \mathbf{A}\,\mathbf{T}\,\mathbf{F}_x + \mathbf{A\Phi T}\,\mathbf{F}_y \tag{1.41}$$

(F span the null-space of $[\mathbf{E}_x \mid \mathbf{E}_y]$).

In the above equation $[\mathbf{E}_x \mid \mathbf{E}_y]$ is an m × 2d matrix, it can be seen as consisting of m vectors in a 2d dimensional space, and the set of all vectors which transform into the zero vector (i.e. which satisfy $[\mathbf{E}_x \mid \mathbf{E}_y]\ x = 0$) is called the null space of A, and it has a dimension 2d-rank $[\mathbf{E}_x \mid \mathbf{E}_y]$ or d. Now if

$$\Psi = -\mathbf{F}_x[\mathbf{F}_y^{-1}] \tag{1.42}$$

then

$$\mathbf{AT}\,\Psi\,\mathbf{T}^{-1} = \mathbf{A\Phi} \tag{1.43}$$

If A is assumed to be a full rank matrix:

$$\mathbf{T}\,\Psi\,\mathbf{T}^{-1} = \Phi \tag{1.44}$$

Thus the eigenvalues of Ψ correspond to the diagonal element of Φ

Summary of the TLS ESPRIT

1) Obtain an estimate of the data covariance matrix $\mathbf{R}_{zz}$, denoted by $\overline{\mathbf{R}}_{zz}$.

2) Perform the eigendecomposition of $\overline{\mathbf{R}}_{zz}$ as $\overline{\mathbf{R}}_{zz} = \mathbf{E}\,\Lambda\,\mathbf{E}$ where

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots \lambda_{2m})$$

and

$$\mathbf{E} = (\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_{2m})$$

3) Estimate the number of sources d.

4) Obtain $\mathbf{E}_s = [\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d]$ and decompose it to

obtain $\begin{vmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{vmatrix}$

5) Compute the eigendecomposition of $\mathbf{E}_{xy}{}^{*}\mathbf{E}_{xy} = \begin{vmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{vmatrix} [\mathbf{E}_x \mid \mathbf{E}_y] = \mathbf{E}\,\Lambda\,\mathbf{E}$

and partition $\mathbf{E}$ into four d × d submatrices

$$\mathbf{E} = \begin{vmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}_{21} & \mathbf{E}_{22} \end{vmatrix}$$

6) Calculate the eigenvalues of $\Psi = -\mathbf{E}_{12}[\mathbf{E}_{22}^{-1}]$

7) Estimate $\theta_k = f^{-1}(\Phi_k)$

$$\theta_k = \mathrm{Sin}^{-1}\{c\,\arg(\phi_k) / (\omega_0 \Delta)\}$$

1.6: IMPROVED TLS ESPRIT

By considering the eigendecomposition of the data matrix $\mathbf{R}_{zz}$ of rank d, following equation can be written.

$$\mathbf{R}_{zz}\mathbf{e}_i = \lambda_i \mathbf{e}_i = \sigma^2 \mathbf{e}_i \qquad i=d+1, \ldots, 2m \tag{1.45}$$

Using the same procedure as in the MUSIC algorithm

$$\overline{\mathbf{A}}\ \mathbf{G} = 0 \tag{1.46}$$

where

$$\mathbf{G} = [\ \mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots\ldots \quad \mathbf{e}_{2m}\ ]$$

Now from the fact that $\overline{\mathbf{A}}$ and $\mathbf{G}$ can be partitioned as

$$\overline{\mathbf{A}} = \begin{vmatrix} \mathbf{A}\Phi \\ \mathbf{A} \end{vmatrix} \quad \text{and} \quad \mathbf{G} = \begin{vmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{vmatrix} \tag{1.47}$$

Hence

$$(\mathbf{A}^H, \Phi^H \mathbf{A}^H) \begin{vmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{vmatrix} = 0 \tag{1.48}$$

or

$$\mathbf{A}^H \mathbf{G}_x + \Phi^H \mathbf{A}^H \mathbf{G}_y = 0$$

$$\Rightarrow \quad \mathbf{A}^H \mathbf{G}_x = -\Phi^H \mathbf{A}^H \mathbf{G}_y$$

$$\Rightarrow \quad \mathbf{G}^H \mathbf{A} = -\mathbf{G}_y^H \Phi \mathbf{A} \tag{1.49}$$

By multiplying both sides of the above equation by **T** defined in Equation(1.39).

$$\mathbf{G}_x^H \mathbf{A}\,\mathbf{T} = -\mathbf{G}_y^H \mathbf{A}\Phi\,\mathbf{T} \tag{1.49a}$$

or

$$\mathbf{G}_x^H \mathbf{E}_x = -\mathbf{G}_y^H \mathbf{E}_y \tag{1.49b}$$

Because $\mathbf{E}_x$ and $\mathbf{E}_y$ span the same subspace, then the objective in the previous TLS algorithm is to find a matrix $\psi \in \mathbf{C}^{d\times d}$ such that

$$\mathbf{E}_x \psi = \mathbf{E}_y. \tag{1.50}$$

The substitution of (1.50) into (1.49b) yields

$$\mathbf{G}_x^H \mathbf{E}_x = -\mathbf{G}_y^H \mathbf{E}_x \psi \tag{1.51}$$

Thus if there exist ψ which transforms $\mathbf{E}_x$ into $\mathbf{E}_y$, this transformation must also transform $-G_y^H E_x$ into $G_x^H E_x$. (Note that $-G_y^H E_x$ and $G_x^H E_x$ span the same subspace as spanned by the columns of E_x or E_y).

In practical situations, where only a finite number of noisy measurements are available, Equations (1.50) and (1.51) can not be satisfied exactly. A criterion for obtaining a suitable estimate of ψ must be formulated. The TLS is a method of fitting that is appropriate in this case because $\mathbf{E}_x$, $\mathbf{E}_y$, $G^H{}_y E_x$, and $G^H{}_x E_x$ are all noisy measurements .

To find a common transformation which satisfies both (1.50) and (1.51) , define

$$H_1 = \begin{vmatrix} E_x \\ -G_y^H E_x \end{vmatrix} \quad \text{and} \quad H_2 = \begin{vmatrix} E_y \\ G_x^H E_x \end{vmatrix} \tag{1.52}$$

thus ψ is given by

$$H_1 \psi = H_2 \tag{1.53}$$

The previous TLS algorithm applied to the model $E_x \psi = E_y$ can be viewed as using m observations (the number of rows of E_x or E_y). By using Equation (1.53), it is easily verified that the number of observations is increased from m to 3m- d . Thus a better estimate of ψ is believed to be achieved, and the algorithm of the improved TLS will be the same as for the TLS with the exception of replacing E_{xy} by $E_{H_1 H_2}$.

However the same solution for Ψ can be achieved by considering instead the d matrices

$$K_i = \begin{vmatrix} H_1^H H_1 & H_1^H h_{2i} \\ h_{2i}^H H_1 & h_{2i}^H h_{2i} \end{vmatrix} \tag{1.54}$$

where h_{2i} is the i^{th} column of the matrix H_2. If λ_{d+1} is the smallest eigenvalues of K_i, then

$$K_i e_{d+1} = \lambda_{d+1} e_{d+1} \tag{1.55}$$

Now by transforming $\mathbf{e}_{d+1}$ into $\begin{vmatrix} X_i \\ -1 \end{vmatrix}$, X_i solves the TLS problem and gives the i^{th} column of Ψ [8].

This transformation is very useful for parallel processing as it avoids the computation of $\mathbf{F}_y^{-1}$ to find Ψ. However this method has a disadvantage as the eigendecomposition of d matrices must be performed at the same time.

Chapter II

PARALLELIZING OF MUSIC/ESPRIT ALGORITHMS

2.1: INTRODUCTION

First step in the development of parallel architecture for any algorithm is to transform it into a computationally efficient algorithm. This is achieved by carefully studying the algorithm and substituting parts of the algorithm with easily computable ones. For example MUSIC and ESPRIT involve the eigendecomposition which can be written in FORTRAN language or can be computed by calling scientific subroutines from commercially available packages. In this chapter efforts are directed to obtain more efficient procedures for computing MUSIC and ESPRIT algorithms and have been explained in the following.

MUSIC and ESPRIT involve intensive matrix eigendecomposition which involve generalized eigenvalue analysis, that is equivalent to the determination of the roots of characteristic polynomial as shown below:

$$P(\lambda) = \det(\mathbf{R}_{xx} - \lambda \mathbf{I})$$

$$= (-1)^n (\lambda^n + C_{n-1}\lambda^{n-1} + \ldots\ldots C_1 \lambda + C_0) \qquad (2.1)$$

The idea of explicitly computing the coefficients of a polynomial, at first glance may appear attractive in view of the fact that operations on a polynomial are easy to perform. However as n increases, the round off errors in computing the coefficients may lead to very large errors in the resulting roots. Generally all the methods used in solving (2.1) are iterative. In fact the data covariance matrix $\mathbf{R}_{xx}$

can be transformed into a diagonal matrix by using the iterative scheme described below.

If $\mathbf{R}_{xx} = \mathbf{R}_1$ is an m ×`m matrix and if $\mathbf{Q}_1$ is an orthogonal matrix, it can be shown that the eigenvalues of $\mathbf{R}_1$ are invariant under a congruent transformation with $\mathbf{Q}_1$.

$$\mathbf{R}_1 \mathbf{X} = \lambda \mathbf{X}$$

$$(\mathbf{Q}_1^H \mathbf{R}_1 \mathbf{Q}_1)(\mathbf{Q}^H \mathbf{X}) = \lambda (\mathbf{Q}^H \mathbf{X}) \qquad (2.2)$$

Thus if (X, λ) is eigenpair of $\mathbf{R}_1$ then $(\mathbf{Q}_1^H \mathbf{X}, \lambda)$ is eigenpair of $\mathbf{Q}_1^H \mathbf{R}_1 \mathbf{Q}_1 = \mathbf{R}_2$, where $\mathbf{Q}^H$ denotes complex conjugate transpose of matrix $\mathbf{Q}$. The same procedure is repeated for $\mathbf{R}_2$ to obtain $\mathbf{R}_3$ and so on. The matrices $\mathbf{Q}_i$'s are determined in such a way that after a certain number of iterations the matrix $\mathbf{R}_1$ converges to a diagonal one.

The evaluation of eigenvalues and eigenvectors are very important in the process of parallelizing the algorithm. Accuracy in the computations of eigenvalues and eigenvectors will also determine the accuracy of the angle of arrival. The DOA computations need to be performed in real time, hence fast evaluation of eigenvalues and eigenvectors are important. The averaging technique used at various sensors produces a Hermetian covariance matrix from which the eigenvalues and eigenvectors can be obtained.

Structured ımplementation suitable to the high resolution DOA estimation is described for both MUSIC and ESPRIT algorithms. Figure 2.1 gives the flow chart of a pipelined arrangement of MUSIC algorithm. The number of sensors depend on

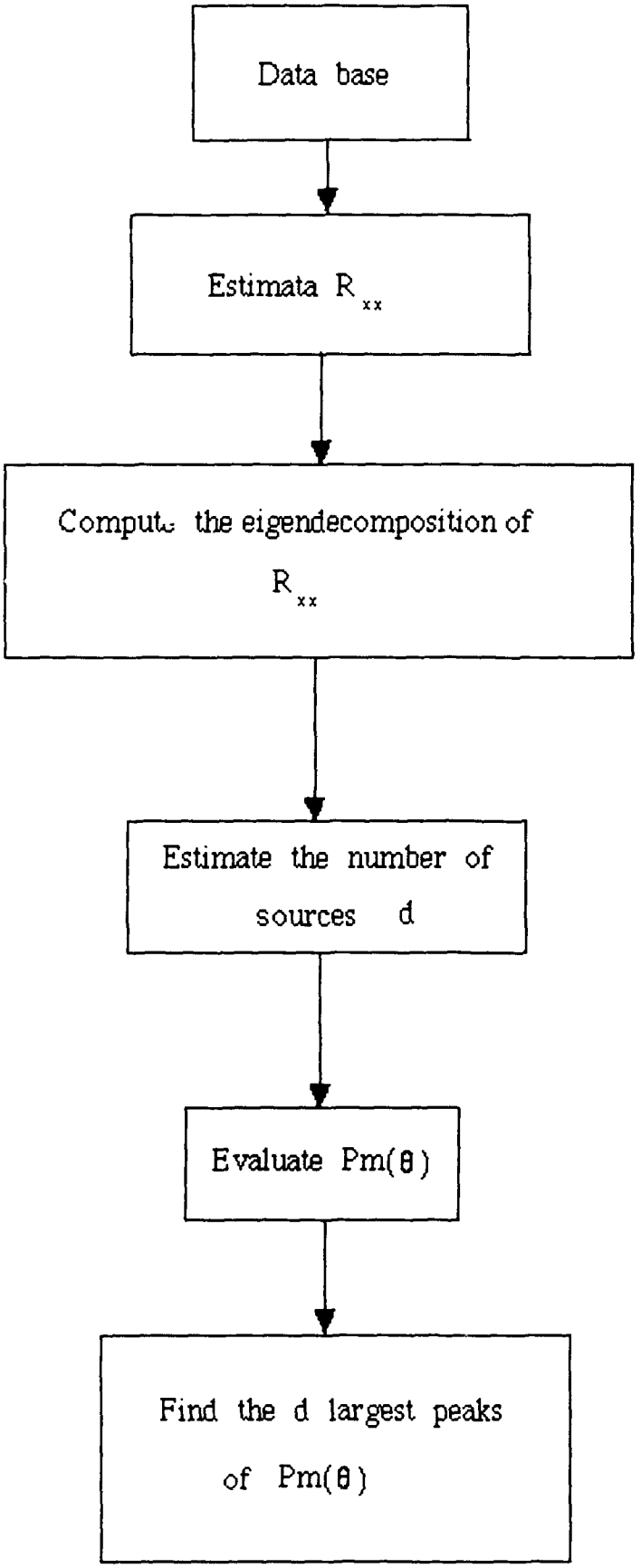


Figure 2.1: Flowchart for MUSIC algorithm

the number of sources need to be located at a given time. The number of sensors should always be greater than the number of sources [4] [5]. Assuming that the number of signals never exceeds seven, a sensor array of eight is considered in the rest of this report.

As shown in Figure 2.1, the data is collected from the sensors. It consists of noisy data vectors **X**(i)'s whose components are x_j , where j and i mean j^{th} sensor and $i^{th,}$ sample respectively. Also a data point at a particular sensor consists of in-phase and quadratic-phase components. The data covariance matrix $\mathbf{R}_{xx}$ is computed using the sampled data.

The eigendecomposition is performed on $\mathbf{R}_{xx}$ and m eigenvalues are derived from which the number of sources is estimated by testing the null hypothesis that the smallest eigenvalue has multiplicity m-d, namely $H_d = \lambda_{d+1} = \lambda_{d+2} = \ldots = \lambda_m$ is tested. The likelihood ratio for the problem, under Guassian assumption, is given by:

$$Q_d = \left(\frac{\prod_{i=d+1}^{m} \lambda_i}{\left(\frac{1}{m-d} \sum_{i=d+1}^{m} \lambda_i \right)^{m-d}} \right)^N$$

where $\lambda_1 > \lambda_2 > \ldots > \lambda_m$

The hypothesis H_d (d = 0, 1,..... m-1) are tested sequentially, and the smallest likelihood ratio is chosen. The final phase of MUSIC is the determination of the

angle of arrival which can be approximated from the elements of array manifold closest to the signal subspace by using the measure function $Pm(\theta)$ described in chapter I. The plotting of $Pm(\theta)$ is computationaly very expensive as it requires the calculation of $Pm(\theta)$ for every angle $\theta \in (-\pi, \pi)$ to find the intersection between the array manifold and the signal subspace given by the d peaks of $Pm(\theta)$.

Similar to pipelined MUSIC flowchart, a flowchart for TLS - ESPRIT algorithm is shown in Figure 2.2. The different steps involved in estimating the DOA using ESPRIT are given. ESPRIT is similar to MUSIC both of which exploit the underlying data model, but it eliminates the computation of $Pm(\theta)$ for every possible θ. However it involves some more eigendecomposition, i.e., the eigendecomposition of $\mathbf{E}^{H}{}_{XY}\,\mathbf{E}_{XY}$ and $-\mathbf{E}_{12}\,\mathbf{E}^{-1}{}_{22}$ and the inversion of matrix $\mathbf{E}_{22}$ as shown in Figure 2.2. Also one should note that the order of data covariance matrix has increased from m to 2m. In the following sections, algorithms for various computational modules required in MUSIC and ESPRIT are developed.

2.2: DATA COVARIANCE MATRIX FORMATION

Once the data has been collected by the sensors the data covariance matrix can be computed using

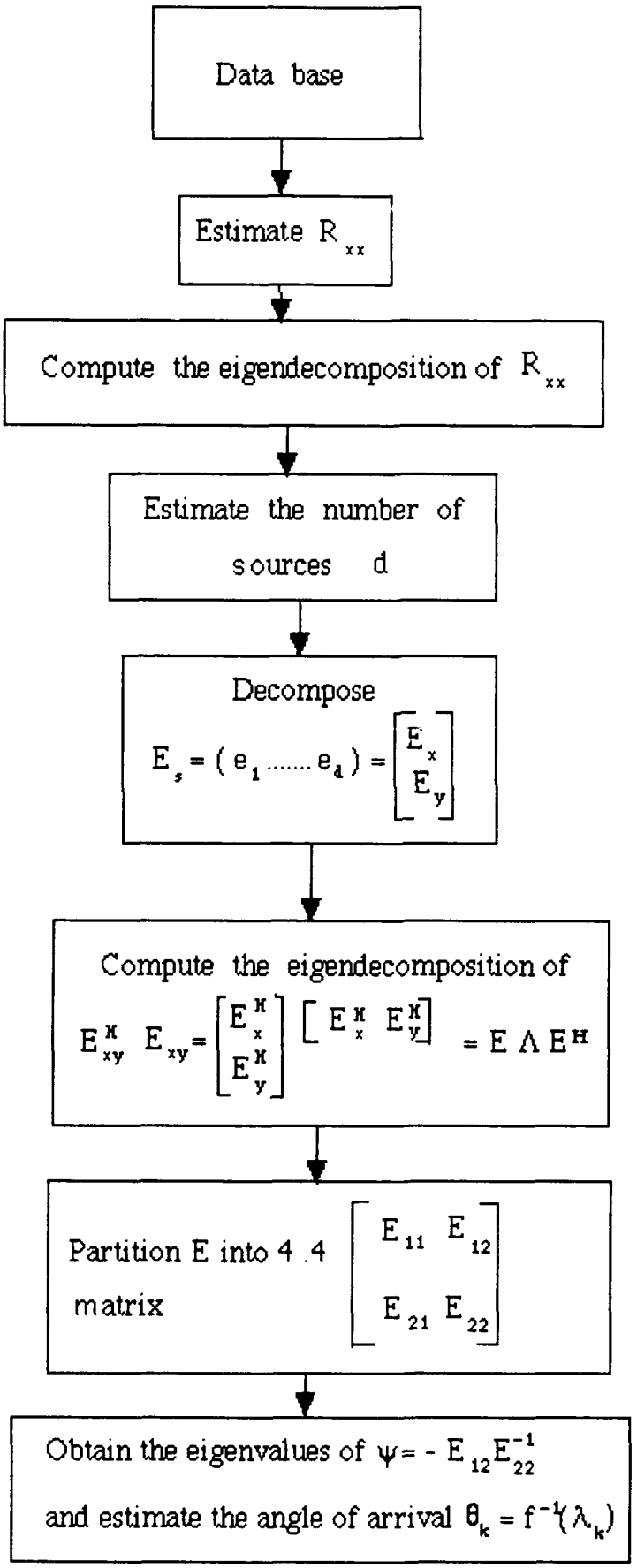


Figure 2.2: Flowchart for pipelined arangement for ESPRIT

$$\mathbf{R}_{xx} = \frac{\sum_{p=1}^{n} x(p).x(p)^*}{n} \qquad (2.3)$$

Where $\mathbf{R}_{xx}$ = Covariance of data matrix

$x(p)$ = Vector of data output from every sensor at p^{th} sample given by $(x_1, x_2, \ldots\ldots x_8)^T$

n = Number of samples

The sampled data obtained from the sensors is used to obtain the data covariance matrix given by equation (2.3). For instance, the element (i,j) of $\mathbf{R}_{xx}$ denoted by $\mathbf{R}_{ij}$ is computed as:

$$\mathbf{R}_{ij} = \frac{\sum_{p=1}^{n} x_i(p) \cdot x_j^*(p)}{n}$$

Since the covariance matrix is Hermetian, the computation of lower triangular matrix of covariance matrix is sufficient to get complete information of the full matrix.

The sequence of transformations described earlier as in equation (2.2) to reduce the data covariance matrix to a diagonal one is one of the most efficient algorithms for determining all the eigenvalues and eigenvectors. It is known as QR algorithm. However when applied to a dense matrix, it is very time consuming, requiring a large number of computations. In order to circumvent this drawback, the data covariance matrix is transformed first to tridiagonal one using Householders transformation. Chen [9] and Dongarra [10] discussed the reduction

of a Hermetian matrix to a tridiagonal matrix using Householders transformation. The eigenvalues of $\mathbf{R}_{xx}$ are invariant under this transformation as the Householder transformation is orthogonal. Hence, the Householders transformation to reduce the dense matrix to a tridiagonal matrix, QR method to reduce tridiagonal matrix to a diagonal one, the power method to compute the angle of arrival are discussed here.

2.3: HOUSEHOLDERS TRANSFORMATION

First the mathematical aspect of Householder transformation is presented, then its parallelization and its hardware implementation is explored. The method of Householder [11], can effectively reduce the bandwidth of data covariance matrix $\mathbf{R}_{xx}$ by transforming it to a tridiagonal matrix **T**. In order to transform the $m \times m$ data matrix into a tridiagonal one, m-2 Householder's transformations (N) are determined such that

$$\mathbf{N}^H \mathbf{R}_{xx} \mathbf{N} = \mathbf{T}$$

where $\mathbf{N} = \mathbf{N}_{m-2}\,\mathbf{N}_{m-3}\cdots\cdots\mathbf{N}_1$.

$$N_k = \begin{bmatrix} I & 0 \\ 0 & \overline{N_k} \end{bmatrix} \begin{matrix} k \\ m-k \end{matrix}$$

$$\begin{matrix} k & m-k \end{matrix}$$

and

$$\overline{\mathbf{N}}_k = \mathbf{I} - \frac{2\,\mathbf{w}\,\mathbf{w}^H}{\mathbf{w}^H\,\mathbf{w}} \qquad \text{with } \mathbf{N}_k \in C^{m\times m} \text{ and } \mathbf{w} \in C^m$$

The matrices $\mathbf{N}_k$ are determined to eliminate the elements above and below the subdiagonals without disturbing any previously zeroed rows and columns. The basic iterative sequence of operations for this transformation method can be stated as

$$\mathbf{R}_1 = \mathbf{R}_{xx}$$

begin
For k=1,2,...,m-2,

$$\mathbf{R}_{k+1} = \mathbf{N}_k^H \mathbf{R}_k \mathbf{N}_k \qquad \text{such that } \mathbf{N}_k^H \mathbf{N}_k = \mathbf{I}$$

end

$$\mathbf{T} = \mathbf{R}_{m-2}$$

For k=1, the transformation can be written as

$$\mathbf{R}_2 = \mathbf{N}_1^H \mathbf{R}_1 \mathbf{N}_1$$

where

$$\mathbf{R}_1 = \begin{array}{c} \begin{matrix} 1 \end{matrix} \\ \begin{matrix} 1 \\ \\ m-1 \end{matrix} \left[\begin{array}{c|c} r_{11} & \mathbf{r}_1^H \\ \hline \mathbf{r}_1 & \overline{R}_1 \end{array}\right] \begin{matrix} 1 \\ \\ m-1 \end{matrix} \\ \begin{matrix} 1 & m-1 \end{matrix} \end{array}$$

Therefore

$$R_2 = \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \bar{N}_1^H \end{array}\right] \left[\begin{array}{c|c} r_{11} & \mathbf{r}_1^H \\ \hline \mathbf{r}_1 & \bar{R}_1 \end{array}\right] \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \bar{N}_1 \end{array}\right]$$

$$R_2 = \left[\begin{array}{c|c} r_{11} & \mathbf{r}_1^H \\ \hline \bar{N}_1^H \mathbf{r}_1 & \bar{R}_1 \end{array}\right] \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \bar{N}_1 \end{array}\right]$$

$$R_2 = \begin{array}{c} 1 \\ 1 \end{array} \left[\begin{array}{c|c} r_{11} & \mathbf{r}_1^H \bar{N}_1 \\ \hline \bar{N}_1^H \mathbf{r}_1 & \bar{N}_1^H \bar{R}_1 \bar{N}_1 \end{array}\right] \begin{array}{c} 1 \\ m-1 \end{array}$$

$$\begin{array}{cc} 1 & m-1 \end{array}$$

In order for the first term $\bar{N}_1^H \mathbf{r}_1$ to be null except for the first element, '**w**' should have the following form

$$\mathbf{w} = \mathbf{r}_1 + \beta \mathbf{e}_1$$

where $\mathbf{e}_1 = (1, 0, 0, 0...)^T$

and $\mathbf{r}_1 = (r_{21}, r_{31}, \ldots\ldots r_{n1})^T$

β is a complex number that is to be determined.

$$\bar{N}_1^H \mathbf{r}_1 = \left\{ \mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H \mathbf{w}} \right\}^H \mathbf{r}_1$$

$$= \left\{ \mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H \mathbf{w}} \right\} \mathbf{r}_1$$

$$= \mathbf{r}_1 - \frac{2(\mathbf{r}_1 + \beta\,\mathbf{e}_1)\,(\mathbf{r}_1^H + \beta^*\,\mathbf{e}_1^T)\,\mathbf{r}_1}{(\mathbf{r}_1^H + \beta^*\mathbf{e}_1^H)\,(\mathbf{r}_1 + \beta\,\mathbf{e}_1)}$$

$$= -\,\beta\,\mathbf{e}_1$$

For this equation to be satisfied, following can be written

$$2\,(\mathbf{r}_1^H + \beta^*\mathbf{e}_1^T)\,\mathbf{r}_1 = (\mathbf{r}_1^H + \beta^*\mathbf{e}_1^T)(\mathbf{r}_1 + \beta\,\mathbf{e}_1)$$

or

$$2\,\mathbf{r}_1^H\,\mathbf{r}_1 + 2\,\beta^*\,\mathbf{e}_1^T\,\mathbf{r}_1 = \mathbf{r}_1^H\,\mathbf{r}_1 + \beta^*\,\mathbf{e}_1^T\,\mathbf{r}_1 + \beta\,\mathbf{r}_1^H\,\mathbf{e}_1 + \beta^*\,\beta$$

One solution is given by

$$\begin{cases} \mathbf{r}_1^H\,\mathbf{r}_1 = \beta^*\,\beta \end{cases} \tag{2.4}$$

$$\begin{cases} \beta^*\,\mathbf{e}_1^T\,\mathbf{r}_1 = \beta\,\mathbf{r}_1^H\,\mathbf{e}_1 \end{cases} \tag{2.5}$$

multiplying both sides of (2.5) by β^*.

$$(\beta^*)^2\,\mathbf{e}_1^T\,\mathbf{r}_1 = \beta^*\,\beta\,\mathbf{r}_1^H\,\mathbf{e}_1 \tag{2.6}$$

Substitution of (2.6) in (2.4) yields

$$(\beta^*)^2 = \frac{\mathbf{r}_1^H\,\mathbf{r}_1\;\mathbf{r}_1^H\,\mathbf{e}_1}{\mathbf{e}_1^T\,\mathbf{r}_1}$$

Using the fact that

$$\mathbf{r}_1^H \mathbf{r}_1 = \left\| \mathbf{r}_1 \right\|^2 = \beta_1^2$$

$$\mathbf{r}_1^H \mathbf{e}_1 = r_{21}^* \text{ and}$$

$$\mathbf{e}_1^T \mathbf{r}_1 = r_{21}$$

This will give

$$(\beta^*)^2 = \beta_1^2 \frac{r_{21}^*}{r_{21}}$$

Multiplying and dividing the right hand side by r_{21}^*

$$(\beta^*)^2 = \beta_1^2 \frac{(r_{21}^*)^2}{r_{21}^* . r_{21}}$$

$$= \beta_1^2 \frac{(r_{21}^*)^2}{|r_{21}|}$$

or

$$\beta^* = \beta_1 \frac{r_{21}^*}{|r_{21}|}$$

$$\Rightarrow \beta = \beta_1 \frac{r_{21}}{|r_{21}|} \qquad (2.7)$$

By choosing β given by (2.7), the first column of $\mathbf{R}_2$ has the form $(r_{11}, -\beta, 0, 0, 0...0)^T$,

and because of the Hermetian property the first row becomes $(r_{11}, -\beta^{*}, 0, 0,...0)$.

In the following and for simplicity, a method is given to calculate the elements of $\mathbf{R}_2$ for the real case, i.e., the data is composed of in-phase components only. The same method may be applied for the complex case where the data is composed of in-phase and quadrature-phase components. Hence for real variables the Householders transformation reduces to:

$$\mathbf{R}_2 = \mathbf{N}_1 \mathbf{R}_1 \mathbf{N}_1$$

$$= \left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} \right] \mathbf{R}_1 \left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} \right]$$

$$= \left[\mathbf{R}_1 - \frac{2\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} . \mathbf{R}_1 \right] \left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} \right]$$

$$= \mathbf{R}_1 - \frac{2\,\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} \mathbf{R}_1 - \mathbf{R}_1 \frac{2\,\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}} + 4 \frac{\mathbf{w}\mathbf{w}^T \mathbf{R}_1 \mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}\,\mathbf{w}^T\mathbf{w}} \qquad (2.8)$$

Let

$$\mathbf{w} = \mathbf{r}_1 + \beta \mathbf{e}_1$$

$$c = \frac{\mathbf{w}^T\mathbf{w}}{2}$$

and,

$$\mathbf{d} = \overline{\mathbf{R}}_1 \mathbf{w}$$

Equation (2.8) can be rewritten as follows:

$$\mathbf{R}_1 = \mathbf{R}_1 - \frac{\mathbf{w}\,\mathbf{d}^T}{c} - \frac{\mathbf{d}\,\mathbf{w}^T}{c} + \frac{\mathbf{w}\,(\mathbf{w}^T\,\mathbf{d})\,\mathbf{w}^T}{c^2}$$

$$= \mathbf{R}_1 - \frac{\mathbf{w}\,\mathbf{d}^T}{c} - \frac{\mathbf{d}\,\mathbf{w}^T}{c} + \frac{\mathbf{w}\,(\mathbf{w}^T\,\mathbf{d})\,\mathbf{w}^T}{2\,.\,c^2} + \frac{\mathbf{w}\,(\mathbf{w}^T\,\mathbf{d})\,\mathbf{w}^T}{2\,.\,c^2}$$

$$= \mathbf{R}_1 - \left(\frac{\mathbf{d}}{c} - \frac{\mathbf{w}\,(\mathbf{w}^T\,\mathbf{d})}{2\,.\,c^2}\right)\mathbf{w}^T - \mathbf{w}\left(\frac{\mathbf{d}^T}{c} - \frac{(\mathbf{w}^T\,\mathbf{d})\mathbf{w}^T}{2\,.\,c^2}\right) \qquad (2.9)$$

Let

$$\mathbf{v}^T = \frac{\mathbf{d}^T}{c} - \frac{\mathbf{d}^T\,\mathbf{w}\,\mathbf{w}^T}{2\,.\,c^2}$$

then

$$\mathbf{v}^T = \frac{\mathbf{d}^T}{c} - \frac{(\mathbf{w}^T\,\mathbf{d})\,\mathbf{w}^T}{2\,.\,c^2}$$

and from the fact that

$$\mathbf{d}^T\,\mathbf{w} = \mathbf{w}^T\,\mathbf{d}$$

$$\mathbf{v} = \frac{\mathbf{d}}{c} - \frac{\mathbf{w}\,(\mathbf{w}^T\,\mathbf{d})}{2c^2}$$

Also let

$$p = \frac{\mathbf{w}^T \mathbf{d}}{2c^2}$$

Therefore

$$\mathbf{v} = \frac{\mathbf{d}}{c} - \mathbf{w}\, p$$

and equation (2.9) becomes

$$\mathbf{R}_2 = \mathbf{R}_1 - \mathbf{w}\,\mathbf{v}^T - \mathbf{v}\,\mathbf{w}^T \qquad (2.10)$$

The choice of above equation is primarily motivated by the interest in the application of parallel processing in computing the elements of the matrix $\mathbf{R}_2$, that is, all the columns of $\mathbf{R}_2$ can be computed in parallel as:

$$\mathbf{R}_{2j} = \mathbf{R}_{1j} - \mathbf{v}_j \mathbf{w} - \mathbf{w}_j \mathbf{v}$$

j=1,2.....8

where $\mathbf{R}_{2j}$ and $\mathbf{R}_{1j}$ are the j^{th} column of $\mathbf{R}_2$ and $\mathbf{R}_1$, and $\mathbf{v}_j$ and $\mathbf{w}_j$ are the j^{th} components of **v** and **w** respectively.

The flowchart for the Householders transformation showing its parallelization and sequence of operations is as given in Figure 2.3. The values of '**w**' and 'c' using the first column of $\mathbf{R}_{xx}$. These values of '**w**' and 'c' are used to compute all the $\mathbf{d}_s$ in parallel. Once the $\mathbf{d}_s$ are evaluated $\mathbf{v}_s$ and 'p' are evaluated. Using all the values evaluated as above new values of the elements of the column are computed. This is the first iteration. After

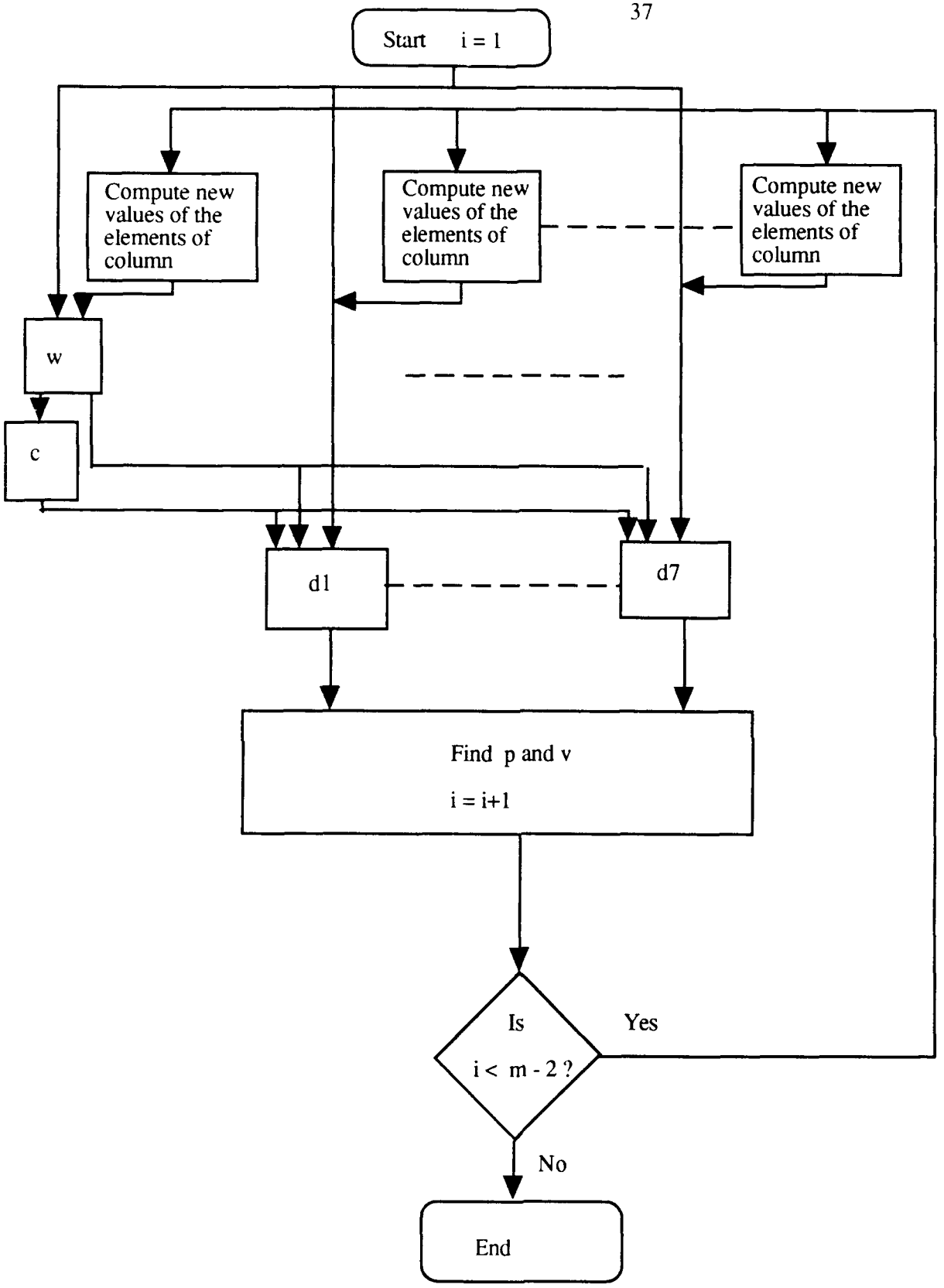


Figure 2.3: Flowchart for parallel Householders transformation

every iteration the counter is incremented. For m - 2 iterations the new values of the columns which are computed are used in feedback loop to perform same operations.

2.4: Q R METHOD

Given the tridiagonal matrix **T** and defining $\mathbf{U} = \mathbf{N}^H$ which is obtained using Householders transformation, QR algorithm may be used to compute eigenvalues and eigenvectors. This is achieved by producing a sequence of transformations based on orthogonal matrices and illustrated by the following algorithm.

$\mathbf{T}_1 = \mathbf{T}$

$\mathbf{U}_1 = \mathbf{N}^H$

begin
for k=1, n

$\mathbf{R}_k = \mathbf{Q}_k^H \mathbf{T}_k$

$\mathbf{T}_{k+1} = \mathbf{R}_k \mathbf{Q}_k$

$\mathbf{U}_{k+1} = \mathbf{Q}^H \mathbf{U}_k$

end

After n iterations **T** will be approximately a diagonal matrix whose diagonal elements are the eigenvalues of the original matrix. The matrix **U** will have rows which are eigenvectors of the original matrix.

An example is given to calculate new values of T for real case. Suppose T_k is obtained at the k^{th}, iteration and we want to calculate T_{k+1} the procedure is as follows:

$$\text{let } Q^T = Q_{n-1}^T Q_{n-2}^T \ldots\ldots Q_1^T$$

where

$$Q_i^{T'} = \begin{bmatrix} 1 & 0 & 0 & & & \\ 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & & \\ & & & \cos & \sin & \\ & & & -\sin & \cos & \\ & & & & & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \end{bmatrix} \begin{matrix} \\ \\ \\ \text{row } i \\ \text{row } i+1 \\ \\ \end{matrix}$$

$$\begin{matrix} & & & & & \text{column } i & \text{column } i+1 \end{matrix}$$

Let the entries of the diagonal elements of tridiagonal matrix be a(m,k) and the entries of subdiagonal elements be b(m,k) where m is row or column number and k is iteration number. The c(m,k) and s(m,k) are sine and cosine that gives angle of rotation of m^{th} and $(m+1)^{th}$ rows and u(m,k) will be the eigenvectors at iteration k.

The tridiagonal matrix **T** is given by

$$T = \begin{bmatrix} a(1,k) & b(2,k) & & & \\ b(2,k) & a(2,k) & b(3,k) & & \\ & b(3,k) & a(3,k) & & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & \ddots \end{bmatrix}$$

From the k^{th} iteration the updated $\mathbf{T}_{k+1}$ using the following equation can be computed.

$$\mathbf{T}_{k+1} = \mathbf{Q}_{n-1}^{T} \mathbf{Q}_{n-2}^{T} \cdots \mathbf{Q}_{1}^{T} \mathbf{T}_{k} \mathbf{Q}_{1} \cdots \mathbf{Q}_{n-2} \mathbf{Q}_{n}$$

However the updated entries b(i, k+1) and a(i, k+1) depend only on the matrices $\mathbf{Q}_{i}$ and $\mathbf{Q}_{i-1}$. They are calculated using

$$\mathbf{Q}_{i}^{T} \mathbf{Q}_{i-1}^{T} \mathbf{T}_{k} \mathbf{Q}_{i-1} \mathbf{Q}_{i} \qquad (2.11)$$

The possibility of parallelizing the QR algorithm is explored base on [15]. Detailed elaboration is given below. To derive the equations for parallel algorithm the following parameters are defined:

cos(i, k) = c1, cos(i-1, k) = c0, sin(i, k)= s1 sin(i-1, k) = s0, a(i, k) = a1 a(i-1, k) = a0, a(i+1, k) = a2, b(i, k) = b1, b(i+1, k) = b2, a(i, k+1) = a3

b(i, k+1) = b3

Substituting the above parameters in equation (2.11)

$$\begin{bmatrix} x & x & x \\ b3 & a3 & x \\ x & x & x \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1 \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0 & s0 & 0 \\ -s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a0 & b1 & 0 \\ b1 & a1 & b2 \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} co & -s0 & 0 \\ s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & -s1 \\ 0 & s1 & c1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1 \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0 & s0 & 0 \\ -s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a0 & b1 & 0 \\ b1 & a1 & b2 \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} c0 & -c1s0 & -s0s1 \\ s0 & c0c1 & -c0s1 \\ 0 & s1 & c1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1 \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0a0+s0b1 & c0b1+s0a1 & s0b2 \\ 0 & -s0b1+c0a1 & c0b2 \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} c0 & -c1s0 & -s0s1 \\ s0 & c0c1 & -c0s1 \\ 0 & s1 & c1 \end{bmatrix}$$

By solving the above matrix the value of b(1, k+1) can be evaluated as

$$b(i, k+1) = (0, \; c1(-s0b1 + c0a1) + s1b2, \quad c0c1b2 + a2s1) \begin{bmatrix} c0 \\ s0 \\ 0 \end{bmatrix}$$

Generalizing for the i^{th} element:

$$b(i, k+1) = s(i-1, k)\{c(i, k)[c(i-1, k)a(i, k) - s(i-1, k)b(i, k)] + s(i, k) \; b(i+1, k)\} \tag{2.12}$$

Let $w = c(i, k)[-s(i-1, k)\, b(i, k) + c(i-1, k)\, a(i, k)] + s(i, k)\, b(i+1, k)$

Substituting w in (2.12)

$$b(i, k+1) = s(i-1, k) \cdot w \tag{2.13}$$

Similarly for a(i, k+1)

$$a(i, k+1) = (0, \; c1(-s0b1 + c0a1) + s1b2, \quad c0c1b2 + a2s1) \begin{bmatrix} -c1c0 \\ c0c1 \\ s1 \end{bmatrix}$$

$$a(i, k+1) = c(i-1, k)c(i, k)\{c(i, k)[c(i-1, k)a(i, k)-s(i-1, k)b(i, k)]+s(i, k)$$
$$b(i+1, k)\} +s(i, k)[c(i-1, k)c(i, k)b(i+1, k)+s(i, k)a(i+1, k) \quad (2.14)$$

Let $v = c(i-1, k)\, c(i, k)\, b(i+1, k) + s(i, k)\, a(i+1, k)$

substituting w and v in equation (2.14)

$$a(i, k+1) = c(i-1, k)\, c(i, k)\, .\, w + s(i, k)\, .\, v \quad (2.15)$$

Similarly values of sin(i, k) and cos(i, k) can be calculated using the following general relation.

$$\cos(i, k) = x(i+1, k)/r \quad (2.16)$$

$$\sin(i, k) = b(i+1, k)/r \quad (2.17)$$

where x(i+1, k) is the updated a(i, k) after $(i-1)^{th}$ rotation and is given by

$$x(i+1, k) = -\sin(i-1, k)b(i, k) + \cos(i-1, k)a(i, k)$$

and

$$r = \text{sqrt}\{[b(i, k)]^2 + x[(i+1, k)]^2\}$$

Using the above values, a pseudocode for the QR algorithm can be written. In this algorithm first parameters are initialized and sine and cosine of the angle of rotation using elements of the matrix as in the equations (2.16) and (2.17) are computed. New values of diagonal elements a_s and subdiagonal elements b_s are then computed using equations (2.13) and (2.15). This process of computing a_s and b_s is repeated as long as all the b_s become 0 or negligible compared to diagonal elements a_s. The pseudocode which performs the QR method is as follows:

```
x(i, k)=0; b(i, k)=u; a(0, k)=0;  b(n+1, k)=0;  c(0, k)=1; s(0, k) = 0;
c(n+1, k)=1;  s(n+1, k)=0;  x1=0;  n=0;
k  = 0
repeat
        for i=1,m
                x(i+1, k)= - s(i-1, k) . b(i, k) + c(i-1, k) . a(i, k)
                r=sqrt{ [b(i, k)]^2+[x(i+1, k)]^2 }

                if r > 0
                        c(i, k) = x(i+1, k)/r
                        s((i, k) = b(i+1, k)/r
                else
                        c(i, k) = 1
                        s(i, k) = 0
                end if
                w=c(i, k) . x(i+1, k)+s(i, k) . b(i+1, k)
                v=c(i-1, k) . c(i, k) . b(i+1, k) + s(i, k) . a(i+1, k)
                b(i, k+1)=s(i-1, k) . w
                a(i, k+1)=c(i-1, k) . c(i, k) . w + s(i, k) . v
                for j=1,m
                        u(i, j, k+1) = c(i, k) . u(i, j, k)+s(i, k) . u(i+1, j, k)
                        u(i+1, j, k+1) = -s(i, k) . u(i, j, k)+c(i, k) . u(i+1, j, k)
                end for
        end for
k = k +1
until sum of squares of b = 0
```

2.5: POWER METHOD

Estimation of the DOA's is performed by using the eigenvectors corresponding to the m - d smallest eigenvalues. It was seen earlier that the direction vectors corresponding to the d sources are orthogonal to the true eigenvectors $e_{d+1}, e_{d+2}, \ldots\ldots e_m$. However, in a practical situation, only estimates of these eigenvectors are available and thus it is expected that the orthogonality criterion does not hold exactly. Instead the projection of these direction vectors on

the subspace spanned by the estimated eigenvectors are used in potential solution, that is the cosine between each of the direction vectors and the estimated eigenvectors $\mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots\ldots \mathbf{e}_m$ is expected to be close to zero.

$$\mathbf{a}^*(\theta_i)\,\mathbf{E}_n \approx 0 \quad i = 1,2,\ldots d$$

$$\mathbf{E}_n = \mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots\ldots \mathbf{e}_m$$

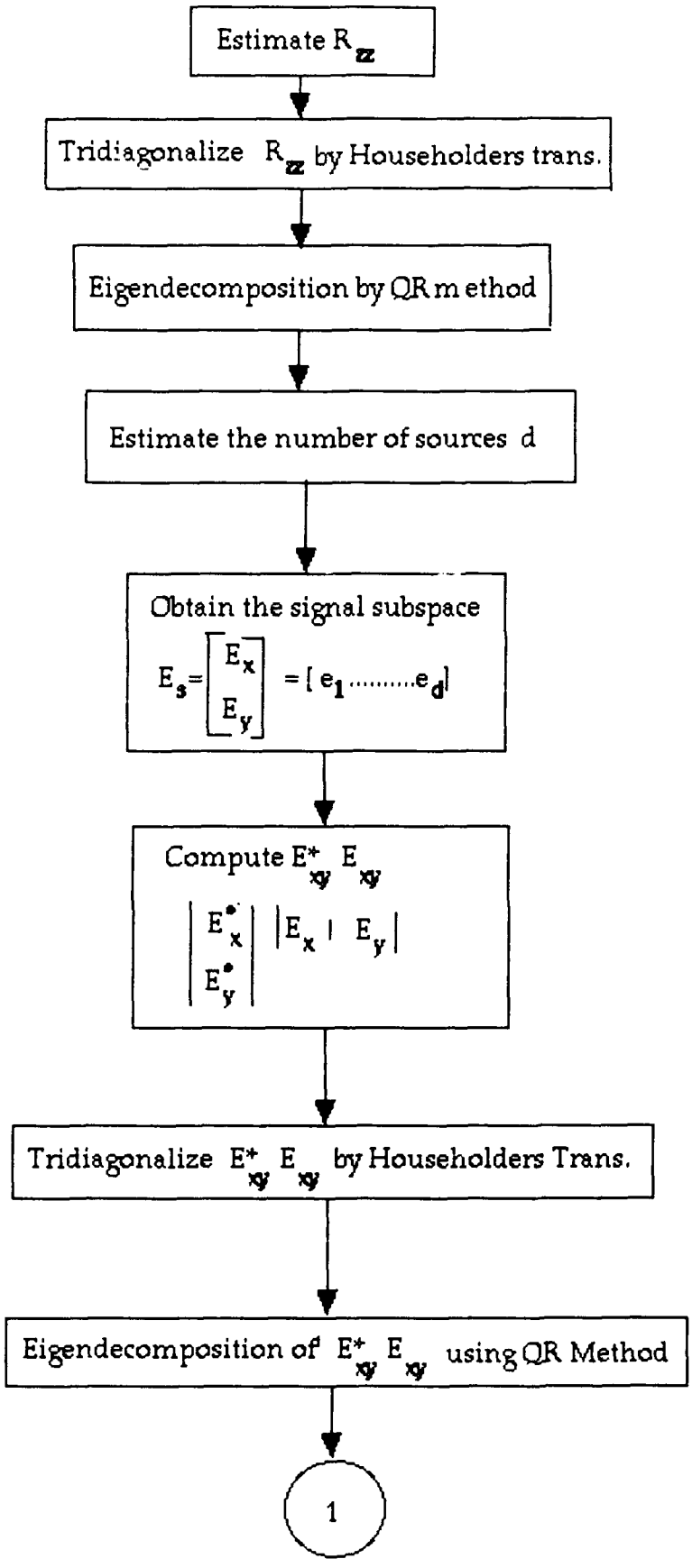
The following function is chosen as one possible measure of closeness of an element of the array manifold to the signal subspace.

$$P_m(\theta) = \frac{1}{\mathbf{a}^*(\theta)\,\mathbf{E}_n \mathbf{E}^*{}_n\,\mathbf{a}(\theta)} \tag{2.18}$$

and the dominant d peaks over $\theta \in (-\pi, \pi)$ are the desired estimate of the directions of arrival.

2.6: COMPUTATIONAL MODULES FOR ESPRIT

A detailed flowchart for the computation of the ESPRIT is shown in Figure 2.4. As shown in the flowchart for pipelined arrangement for ESPRIT most of the operations are similar to MUSIC algorithm. The design of additional required computation modules for ESPRIT such as multiplication of matrices and inversion of matrices are in progress.


Estimate R_{zz}
Tridiagonalize R_{zz} by Householders trans.
Eigendecomposition by QR m ethod
Estimate the number of sources d
Obtain the signal subspace
$E_s = \begin{bmatrix} E_x \\ E_y \end{bmatrix} = [e_1 \ldots\ldots e_d]$
Compute $E^*_{xy}\ E_{xy}$
$\begin{vmatrix} E^*_x \\ E^*_y \end{vmatrix}\ \begin{vmatrix} E_x & E_y \end{vmatrix}$
Tridiagonalize $E^*_{xy}\ E_{xy}$ by Householders Trans.
Eigendecomposition of $E^*_{xy}\ E_{xy}$ using QR Method
1

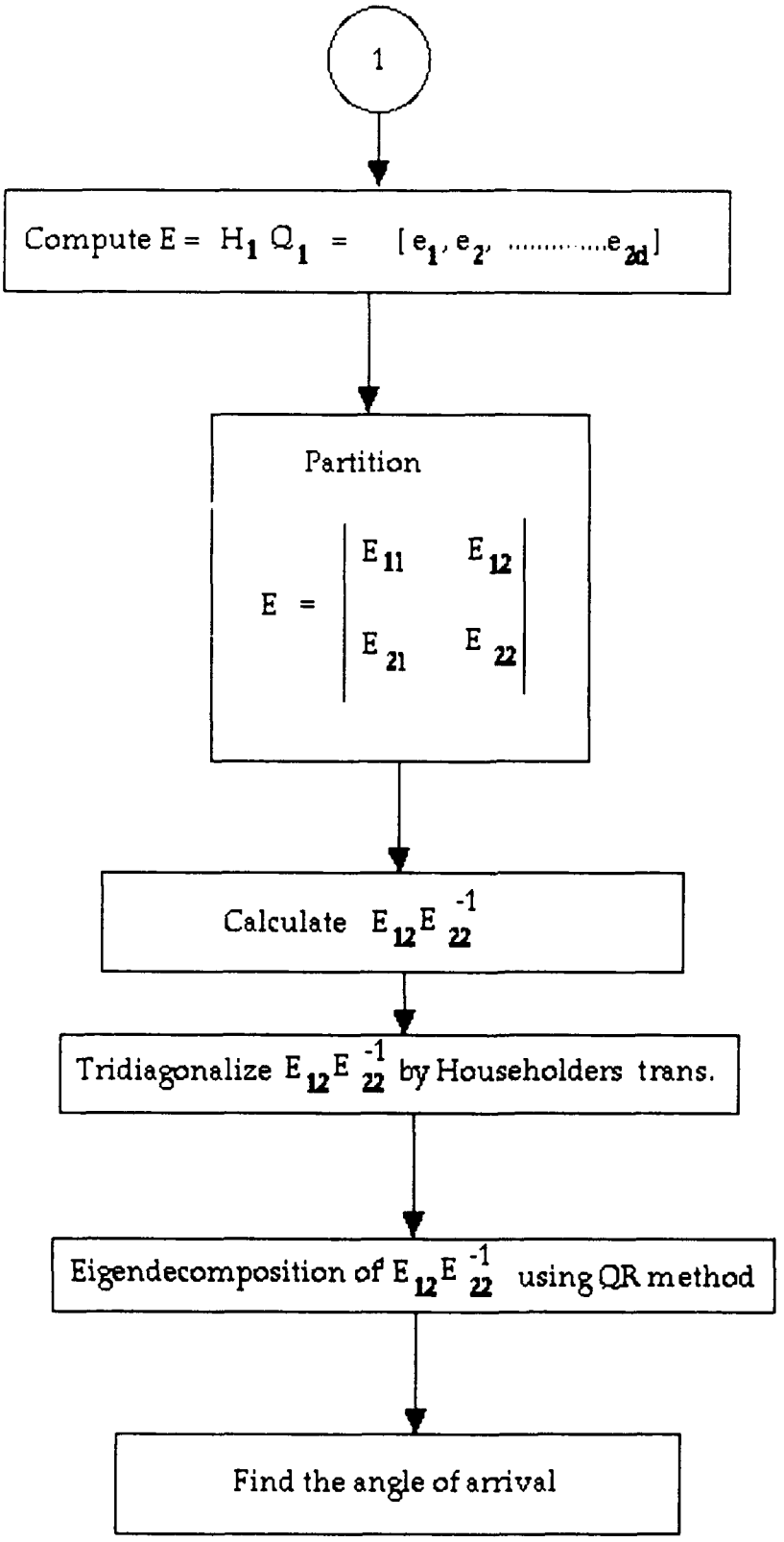


Figure 2.4: Detailed flowchart of ESPRIT Algorithm

Chapter III

HARDWARE IMPLEMENTATION

3.1: INTRODUCTION

With the advances in the area of VLSI it is now possible to design special purpose hardware for the implementation of various real time algorithms. The customized hardware has two main advantages as listed below.

1) The given algorithm is executed at a high speed.
2) Cost and size of the hardware will be lower than the cost of a general purpose computer.

These advantages have led many researchers to probe into the possibility of designing special purpose hardware. The development of special purpose hardware will need to exploit pipeline, parallel and distributed processing approaches to achieve high throughput rates. Hence in this section we present the first step in the development of a dedicated chip set to support MUSIC and ESPRIT algorithm which has been explained in previous sections.

There are many architectures such as systolic array, SIMD Cordic Processors and MIMD which can be used for parallel implementation. An appropriate structure which can exploit maximum parallelization to reduce the computation time will be selected for real time implementation for the particular application.

3.2: LITERATURE SEARCH

Various papers pertaining to parallelization of Householders and QR algorithms were reviewed. C.F.T. Tang et al [12] and K.J.R. Liu [13] proposed architecture for complex Householder transformations for triangularization of the matrix. In their architecture they used single column with the number of processors equal to the number of columns of the matrix. Each processor performs operation on each column. After each iteration the values of each column are fed back to the same processors. But their architecture is proposed to perform triangularization of the given matrix whereas we are interested in tridiagonalization of the covariance matrix.

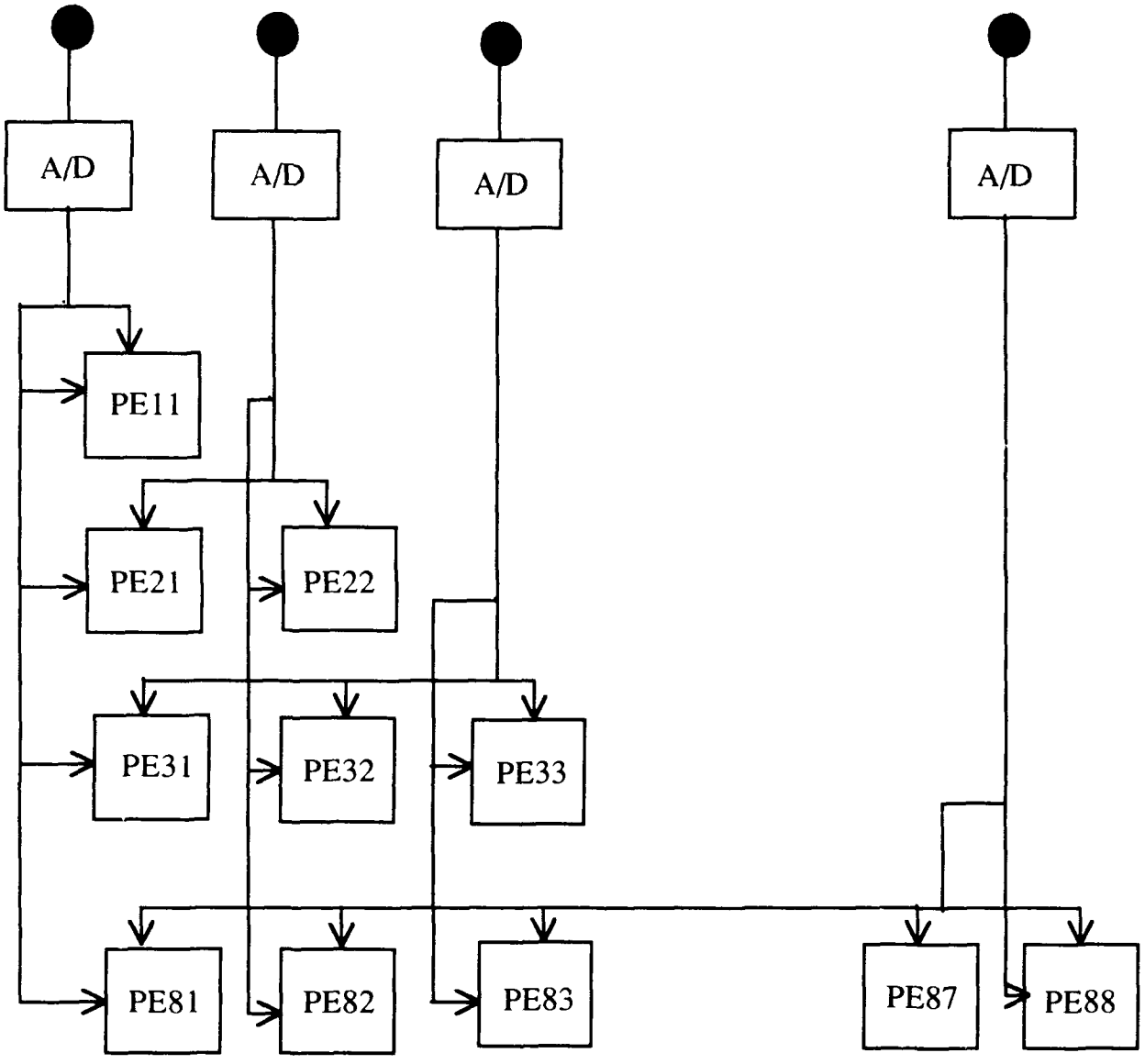
QR method for the tridiagonal matrix is implemented by W. Phillips [15]. In his architecture, rectangular systolic array is used in which each processor performs single iteration. When the first iteration is performed on the m th row by the k th processor, the second iteration is performed on the (m-1) th row by the (k-1) th processor and so on. But the disadvantage in this approach is that the number of processors is dependent on the the number of iterations, i.e., if 5 iterations are required then 5 processors are required. But the exact number of iterations is not known which leads to the uncertainity of the required number of processors.

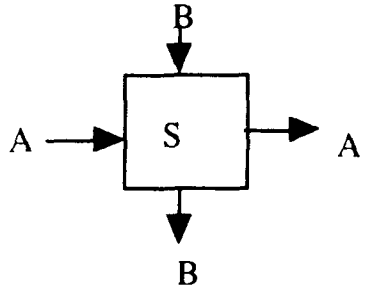
K.J.R.Liu [16] has proposed another kind of approach in which a systolic array arranged in a matrix form is used. The number of processors is equal to the number of elements of the matrix. During first step, the matrix Q is found. Then new values of matrix A are then calculated using Q. Convergence for all the elements of the matrix other than the diagonal elements is checked. If all the elements of the matrix other than the diagonal elements are not equal to zero then the same systolic array

is used for the next iteration. These iterative computations are used until all the elements of the matrix except the diagonal elements converge to zero. The obvious advantage is that the same set of processors can be used for all the iterations. But the drawback is that this architecture is proposed for the evaluation of eigenvalues on the dense matrix. In the next sub-section systolic architecture for the previously developed parallel algorithms for the computations of covariance matrices, householders transformation and QR method is presented.

3.3: SYSTOLIC ARCHITECTURE FOR FORMATION OF DATA COVARIANCE MATRIX.

The parallel computation of the data covariance matrix is performed using systolic architecture. As stated earlier the covariance matrix is Hermetian and computation of lower triangular elements of the covariance matrix is sufficient to get the information for the entire matrix. Since there are 36 elements in the lower triangular 8×8 matrix, systolic architecture will have 36 processors. Here a triangular arrangement of the systolic array with global routing is considered as shown in the Figure 3.1. Each processor is numbered as P(mn) where m is the row number and n is the column number. The sampled data from the $i^{th,}$ sensor is sent to the i^{th} row and the i^{th} column simultaneously. For example the sampled data from the 3^{rd} sensor is sent to all the processors in the third row and the third column. Each processor performs multiplication and addition of two sampled data in parallel in all the processors for every clock cycle . Since there are 36 processors, 36 multiplications and 36 additions are performed simultaneously. Each processor has






```
S = 0
do i=1,p
S = S + A . B
end do
S = S ÷ p
```

Figure 3.1: Systolic architecture for computation of covariance matrix

a memory to store the product of multiplication which is added to the product obtained during the next data cycle. Once the operations of multiplication and addition for all the sampled data in all the processors is performed, the stored data in each processor is then divided by the number of samples in all the processors in parallel. The resulting output are used to form the data covariance matrix R_{xx}.

3.4: SYSTOLIC ARCHITECTURE FOR HOUSEHOLDER TRANSFORMATIONS

The previously computed covariance matrix is a dense matrix. Given a dense matrix it needs to be reduced to a tridiagonal matrix using the Householder transformations. As seen from the flow chart in Figure 2.3 and equation (2.10) the determination of all the d_s and the new elements of the columns of the matrix can be computed in parallel. A systolic architecture is proposed for the computation of tridiagonal matrix. Thus this algorithm can be mapped on a systolic architecture with the number of processors equal to m+1, where m is the order of the matrix. Systolic arrangement for 8×8 matrix is shown in the Figure 3.2. The columns of the matrix are sent to each processor in a pipelined fashion in reverse order such that the last element of the column becomes the first element of the column. The Processor PE1 is used to find the **w** and c required by other processors to find the d_s. Processors PE2, PE3... PE8 are used to find d_s using the value of **w** and c found in the first processor. All the d_s are evaluated in parallel and are sent to the processor PE9. Processors PE2, PE3... PE8 has the memory to store all the w_s, d_s and the elements of that column of the matrix. The processor PE9 is exclusively used for the determination of v_s using d_s and w_s. The v_s are then routed back to all the processors. The processors PE2, PE3... PE8 uses **w**, **d** and **v** to find the new values of the elements of the columns in parallel. At the same time the

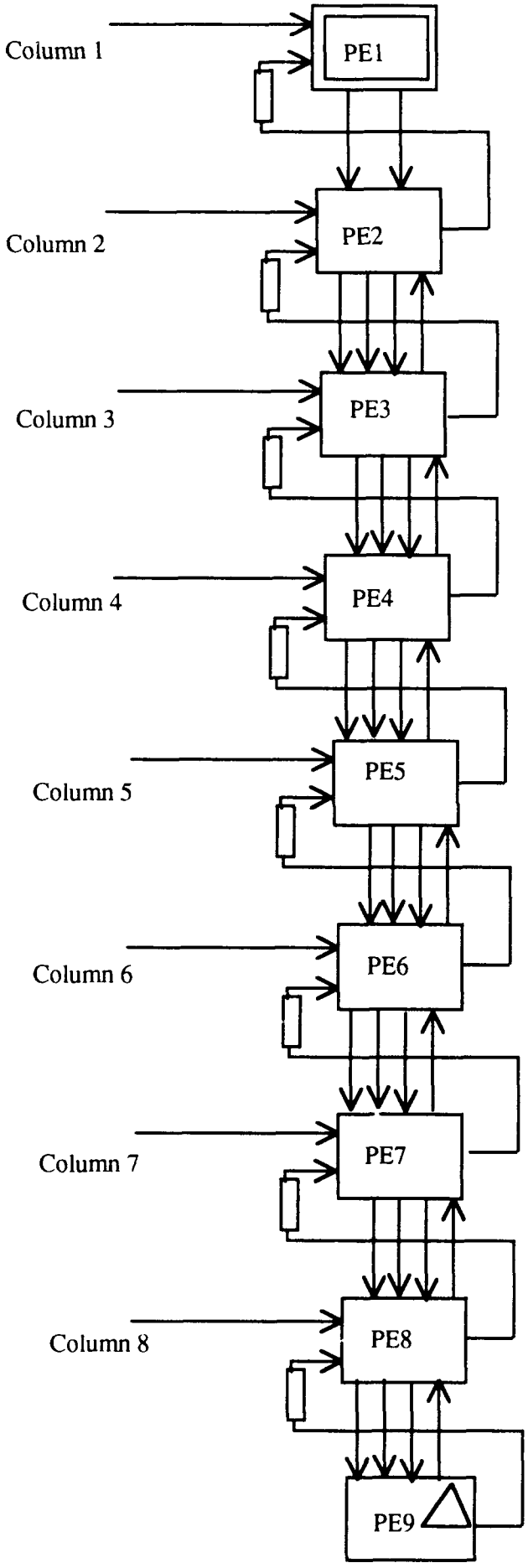
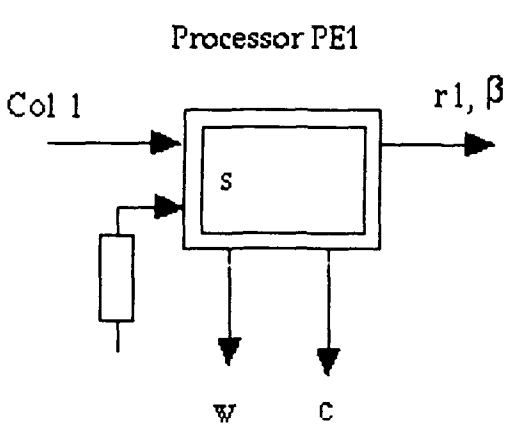


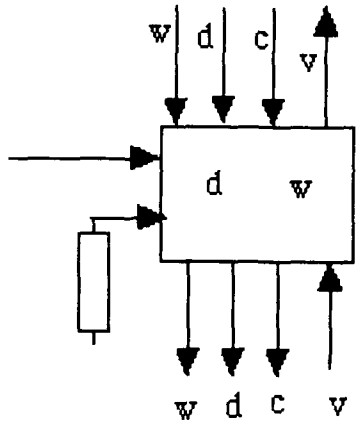
Figure 3.2: Systolic architecture for Householders transformation

first processor is used for the evaluation of β. The first element of the first column and β are the output of the first iteration which are used as input for evaluation of eigenvalues using QR method. The counter is used to set the number of iterations to m-2. For m-2 times, the intermediate results are used in feedback loop and the same set of processors are used repeatedly. The feedback loop has a FIFO memory to temporally store all the elements of the column until operations on previous iteration are completed. For the first iteration operations on 8×8 matrix are performed hence all the processors are utilized. For the second iteration operation on 7×7 matrix are performed. Now the first column of the matrix is already computed; therefore new elements of the second column are fed back to PE1, elements of the third column are fed back to PE2 and so on. Thus for the second iteration PE8 does not have any column to work on and is used only for routing purposes. All other processors perform same operation as in the first iteration, but the elements of each column are reduced by one element. Thus for every new iteration the columns and the elements of the columns goes on reducing. Various operations performed by different processors are given in Figure 3.3.

3.5: SYSTOLIC ARCHITECTURE FOR QR METHOD

The tridiagonal matrix as obtained from the Householders transformations is reduced to a diagonal matrix using the QR method as discussed earlier. The QR algorithm as explained in the pseudocode is described again in the form of parallel flowchart of Figure 3.4. The systolic architecture uses 2*(m+1) processors where 'm' is the number of rows or columns of the tridiagonal matrix. Systolic architecture for 8×8 matrix is shown in Figure 3.5. Each column has (m+1) processors. Two kinds of processors are used. The processors PE1 and PE2 are used to compute the eigenvalues and all the other processors are used to find the eigenvectors of the

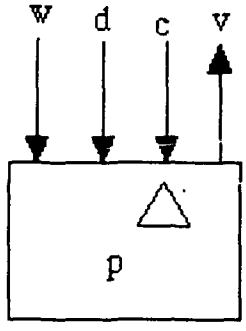





```
i=1
s=0
do 1=8,i+1,-1
   w(1-1)=r(1)
   s = s + w(1-1)*w(1-1)
end do
β= sqrt(s)
w(i)=r(i+1) + β
c= s+ r(i+1).β
i=i+1
```

```
d=0
do 1 = 8,i+1,-1
  d=d+r(1)*w(1-1)
end do
is v in ?
do 1=8,i+1,-1
   r(21)=r(11)-v(1)w-w(1)v
end do
```

Processors PE2,PE3,..,PE8



Processor PE9

```
p=0
do L = 8,i+1,-1
   p = p+w(L-1)*r(L)
end do
p=p/2(c*c)
do L 8,i+1,-1
v(L-1) = d-w(L-1)*p
end do
```

Figure 3.3: Operation performed by different processors.

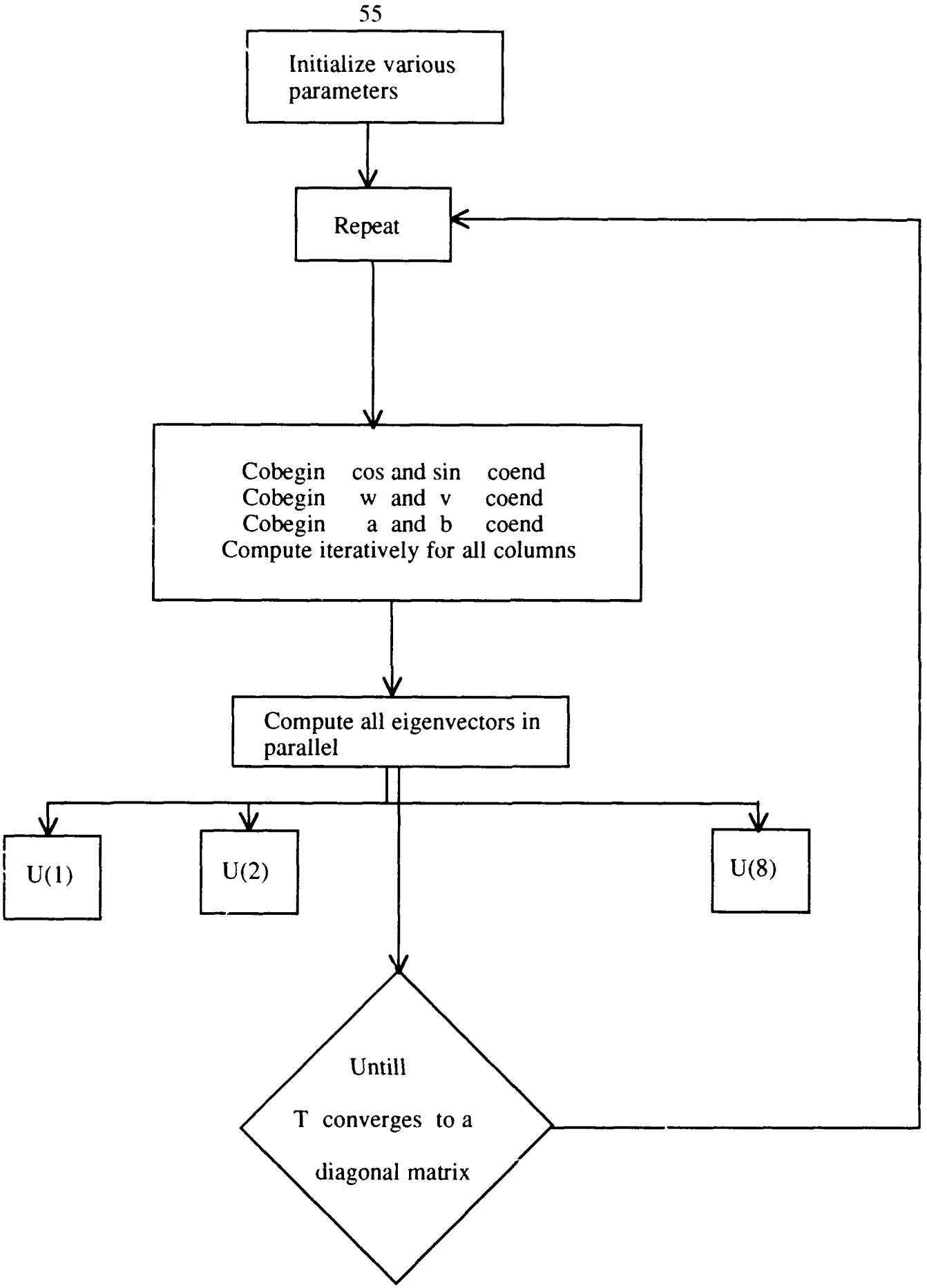


Figure 3.4: Parallel Flowchart for QR algorithm

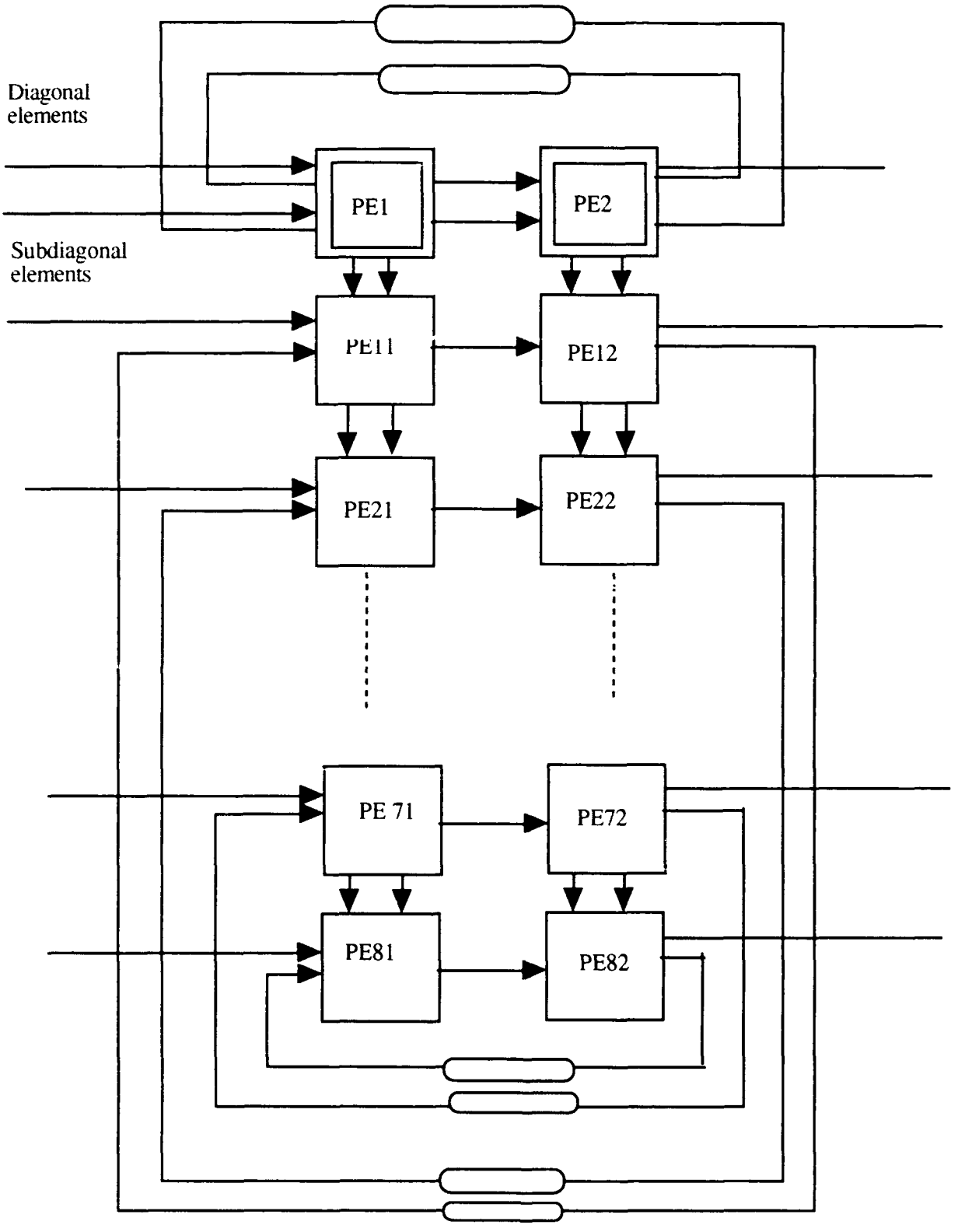


Figure 3.5: Systolic architecture for QR method

tridiagonal matrix. The diagonal and subdiagonal elements of the tridiagonal matrix from the Householders transformation is pipelined into the first processor PE1 which performs the first iteration. The new values of sine and cosine of rotation a_s and b_s are computed. The values of sine and cosine are sent in pipelined fashion to all processors in that column. These processors PE11, PE21....PE81 find the eigenvector of the tridiagonal matrix. The new values of a_s and b_s computed in the first processor are sent to the processor PE2 which performs the second iteration to find the eigenvalue, performs the same operation as performed by PE1. The processors PE12, PE22...PE82 find the eigenvectors for the second iteration. The processor PE2 also performs the test for the convergence. If the sum of squares of the subdiagonal elements is not nearly equal to zero then the control will go into the feedback loop to repeat the computations. This processor is repeated as long as only the diagonal elements are left which form the eigenvalues. Various operations performed by the processor are given in Figure 3.6.

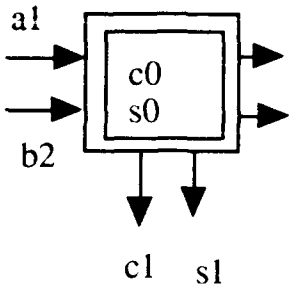
3.6: HARDWARE IMPLEMENTATION OF POWER METHOD

Once the eigenvectors have been computed, the value of eigenvector are utilized to calculate the power as given by equation given in chapter II and is repeated below:

$$P_m(\theta) = \frac{1}{a^*(\theta)\, E_n E^*_n\, a(\theta)}$$

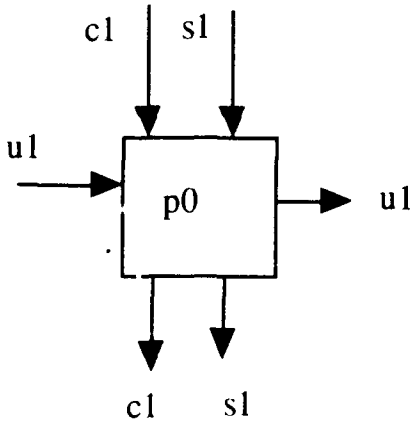
As seen from this equation the evaluation of $a^*(\theta)\, E_n E^*_n\, a(\theta)$ requires squaring of the product of rowvector of $a(\theta)$ and the eigenvector matrix E_n. Hence the product

Processor PE1, PE2




```
x2 = -s1.b2 + c1 a1
r = sqrt(x2.x2+b2.b2)
if x > 0
    c1 = x2/r
    s1 = b2/r
else
    c1 = 1
    s1 = 0
w = c1.x2+s1.b2
v = c0.c1.b2 +s1.a2
a1 = c0.c1.w+ s1.v
b1 = s1.w
```

Processors PE11,PE12,. PE81,PE82




```
c1 = c0   s1 = s0
a1 = c1x1 = s1.a0
p1 = c1.a0- s1.p0
```

Figure 3.6: Operation performed by processors

of $\mathbf{a}(\theta)$ and $\mathbf{E}_n$ is evaluated and then the product obtained is squared and accumulated for all the values in the array manifold. The hardware design is shown in the Figure 3.7. It consists of a set of 8 multipliers to find the product of $\mathbf{a}(\theta)$ and $\mathbf{E}_n$ in parallel. The product obtained is then summed using adders. The real and imaginary and are squared and added again. The evaluation $E^*_n a(\theta)$ is similar to the evaluation of $a^*(\theta)\, E_n$ and hence the product of $a^*(\theta)\, E_n$ is squared and added. The angle of arrival is thus calculated for different values of the angle. The value of $P_m(\theta)$ is different for different angle. The angle for which $P_m(\theta)$ is maximum is the angle of arrival giving the angle of arrival.

3.7: HARDWARE BLOCK DIAGRAM OF MUSIC AND ESPRIT ALGORITHMS

The hardware block diagram for the MUSIC Algorithm is shown in Figure 3.8. As seen in this Figure, the data collected from the sensors is utilized to form the covariance matrix. The eigendecomposition is performed using Householders transformation and QR method. The Eigenvalues are used to find the number of sources and finally using the eigenvectors in Power method we find the angle of arrival. The Hardware block diagram for ESPRIT algorithm is shown in Figure 3.9. It is similar to the MUSIC algorithm but instead of power method of MUSIC some more computations are performed to evaluate the angle of arrival in case of FSPRIT.

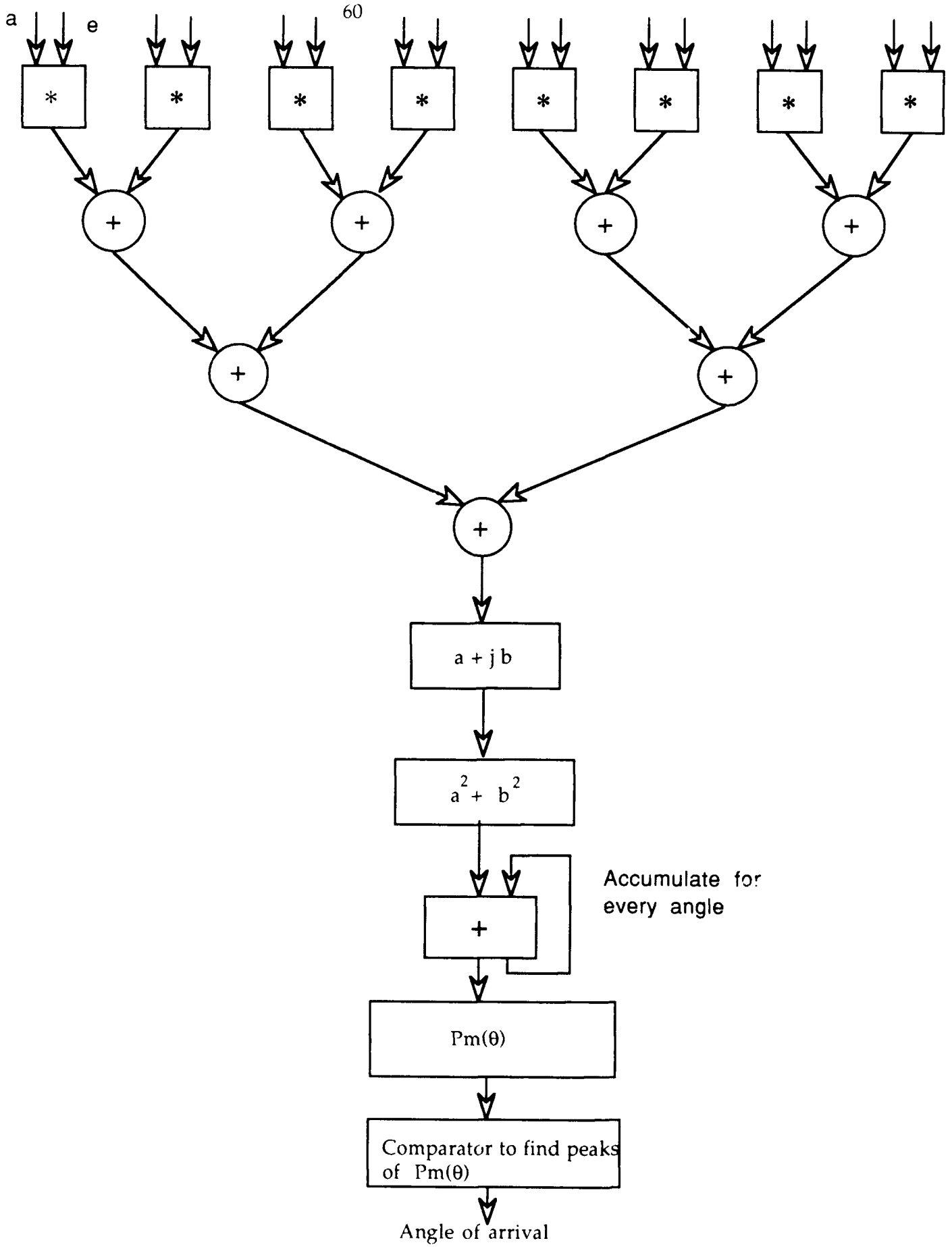


Figure 3.7: Flowchart for power method

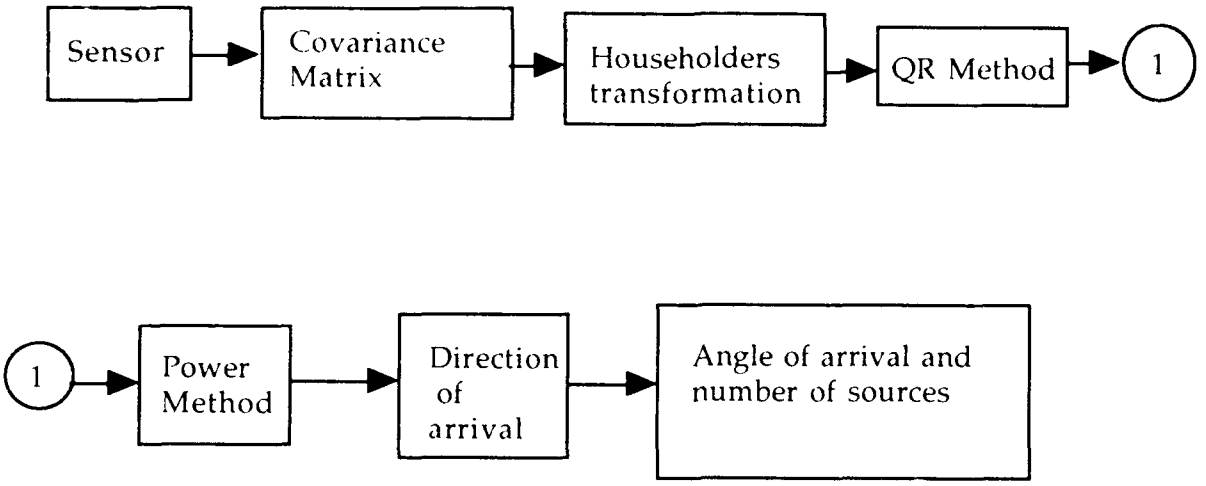


Figure 3.8: Hardware block diagram for MUSIC algorithm

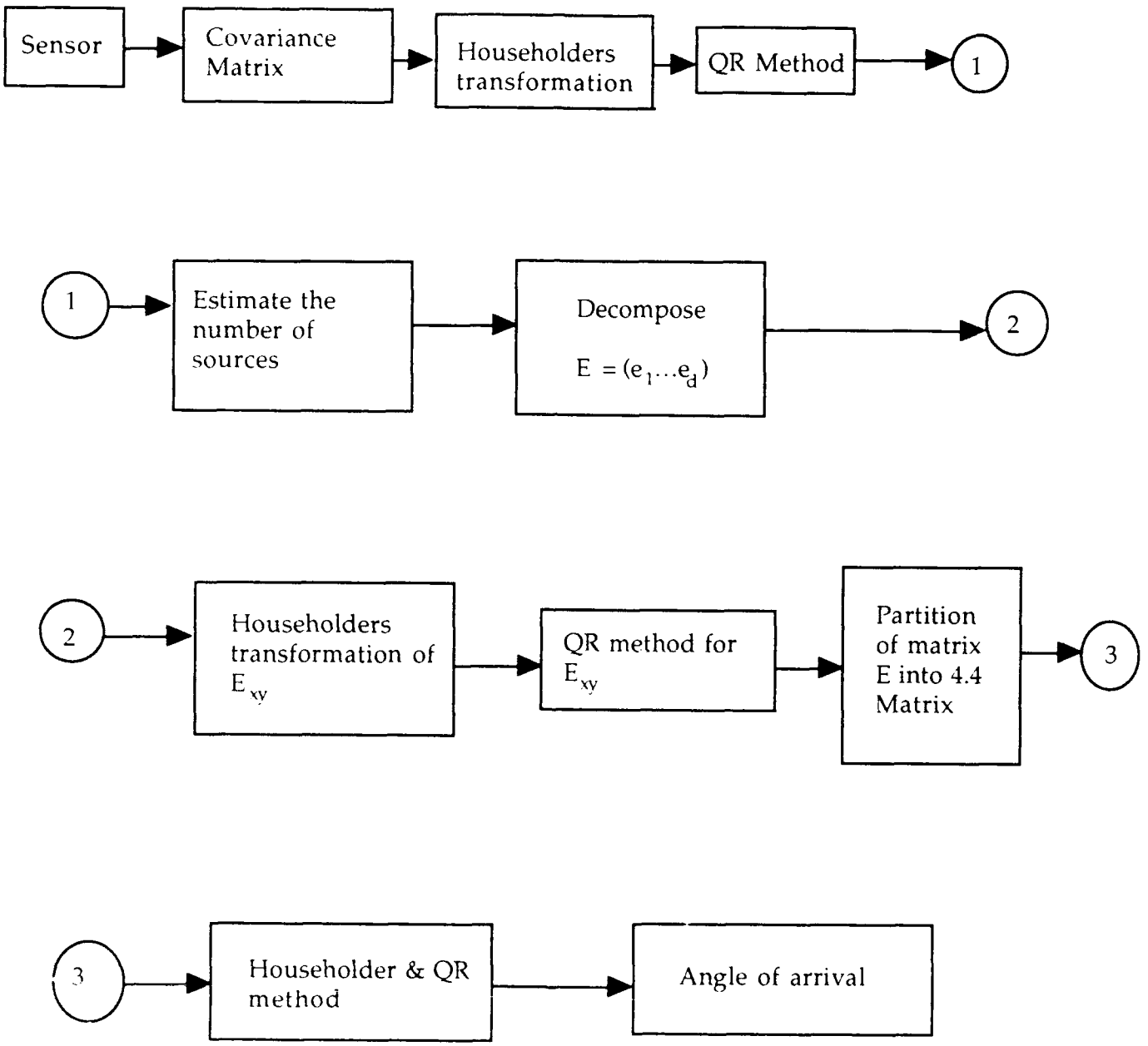


Figure 3.9: Hardware block diagram of ESPRIT algorithm.

Chapter IV

THE CORDIC PROCESSOR APPROACH

4.1: INTRODUCTION

The COORDINATE ROTATION DIGITAL COMPUTER (CORDIC) technique introduced by Volder [17] is highly suitable for the efficient computation of elementary functions like multiplication, division, trigonometric functions, logarithms, exponentials, square root and vector rotation. This technique has been proposed by number of researchers [17-25] for real time digital signal processing applications. Basically, the CORDIC algorithm consists of a set of iterative plane rotations in a linear, circular or hyperbolic coordinate system, depending on which function is to be calculated. The only operations required for the above functions are shifting, adding, subtracting and the use of look up tables as described by Volder [17]. In terms of hardware complexity, CORDIC processing elements are quite simple and consume less chip area. Here we present an alternate method for the computation of eigendecomposition on an array of CORDIC processors. But first the operating principles of the CORDIC processors are described.

4.2: THE CORDIC ALGORITHM:

Consider a plane coordinate system where the radius R and the angle θ of a vector P(x,y) as defined by Walther [18] and Ahmed [23]

$$R = \sqrt{x^2 + m y^2} \tag{4.1}$$

$$\theta = \frac{1}{\sqrt{m}} \tan^{-1}(y\sqrt{m} / x) \tag{4.2}$$

and m = 1 for circular coordinate systems,

m = 0 for linear coordinate systems,

m = -1 for hyperbolic coordinate systems.

Note that for m = -1, Equation (4.2) defines a complex quantity, which can be simplified by rewriting Equation (4.2) in terms of hyperbolic trigonometric functions [18].

After one rotation, the new vector $P_{i+1} = (x_{i+1},y_{i+1})$, as shown in Figure 4.1 obtained from $P_i = (x_i,y_i)$ can be expressed as

$$x_{i+1} = x_i - m\,\sigma_i\,\delta_i\,y_i \tag{4.3}$$

and

$$y_{i+1} = y_i + \sigma_i\,\delta_i\,x_i \tag{4.4}$$

where,

σ_i= +1 for counter clockwise direction of rotation and

σ_i= -1 for clockwise direction of rotation.

δ_i= Integral power of the machine radix given by

$2^{-(i-2)}$ where 'i' is the iteration number of the current CORDIC iteration.

m = Parameter for the coordinate system.

After 'n' iterations, the new radial and angular components of the vector P are given by

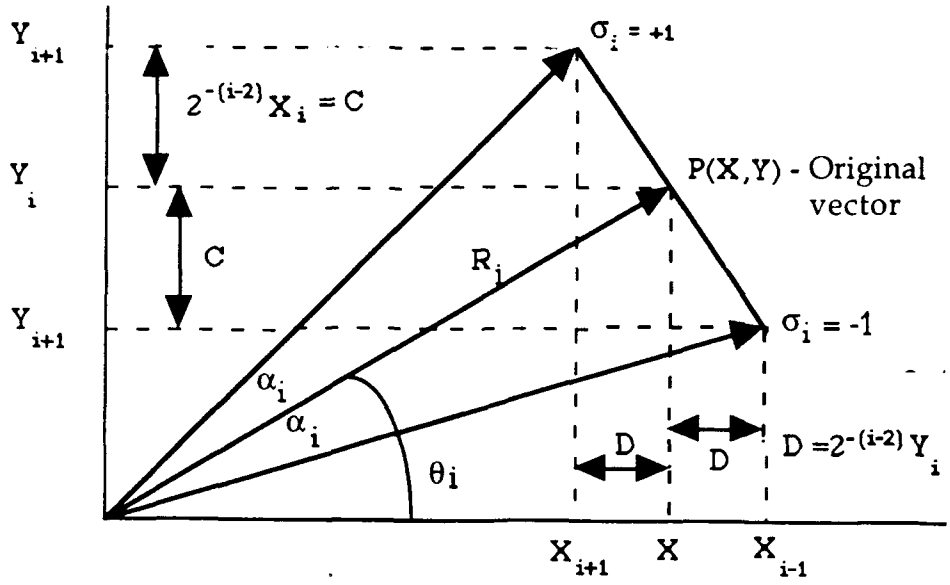


Figure 4.1: Original vector P and the same vector after rotation by $\pm\alpha$ [17].

$$\theta_n = \theta_0 + \alpha \tag{4.5}$$

and

$$R_n = R_0 * K \tag{4.6}$$

where

α and K are given by the following relations [23]

$$\alpha = \sum_{i=0}^{n-1} \sigma_i \alpha_i$$

$$= \sum_{i=0}^{n-1} (\sigma_i / \sqrt{m}) \tan^{-1}(\delta_i \sqrt{m}) \tag{4.7}$$

$$K = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \sqrt{1 + m\,\delta_i^2} \tag{4.8}$$

where,

$\alpha_i = (1/\sqrt{m})\tan^{-1}(\sqrt{m}\ \delta_i)$

and

$K_i = \sqrt{1+ m\,\delta_i^2}$

For n = 24 and m=1, the value of K will be 1.646760255

A third variable 'Z' is provided to keep track of the amount of the angle variations;

$$Z_{i+1} = Z_i + \alpha_i \qquad (4.9)$$

Solving the set of difference Equations (4.3), (4.4), and (4.9), for 'n' iterations [18] gives

$$x_n = K\{x_0 \cos(\alpha\sqrt{m}) + y_0\sqrt{m}\ \sin(\alpha\sqrt{m})\} \qquad (4.10)$$

$$y_n = K\{-x_0\sqrt{m}\ \sin(\alpha\sqrt{m}) + y_0 \cos(\alpha\sqrt{m})\} \qquad (4.11)$$

$$z_n = z_0 + \alpha \qquad (4.12)$$

These relations are summarized in Figure 4.2.

The scale factor K which is inadvertently generated by the CORDIC cycles is not useful and is unwanted at the output. This K can be normalized to unity as suggested by many authors. It was suggested by Haviland and Tuszynski [19] to insert scaling cycles into the CORDIC iteration which scale the magnitude of the vector being rotated by the δ_i used in the CORDIC iteration preceeding the current iteration. For 'n' CORDIC iterations, approximately n/2 scaling cycles are needed in order to cause the scale factors to converge to unity, thus imposing

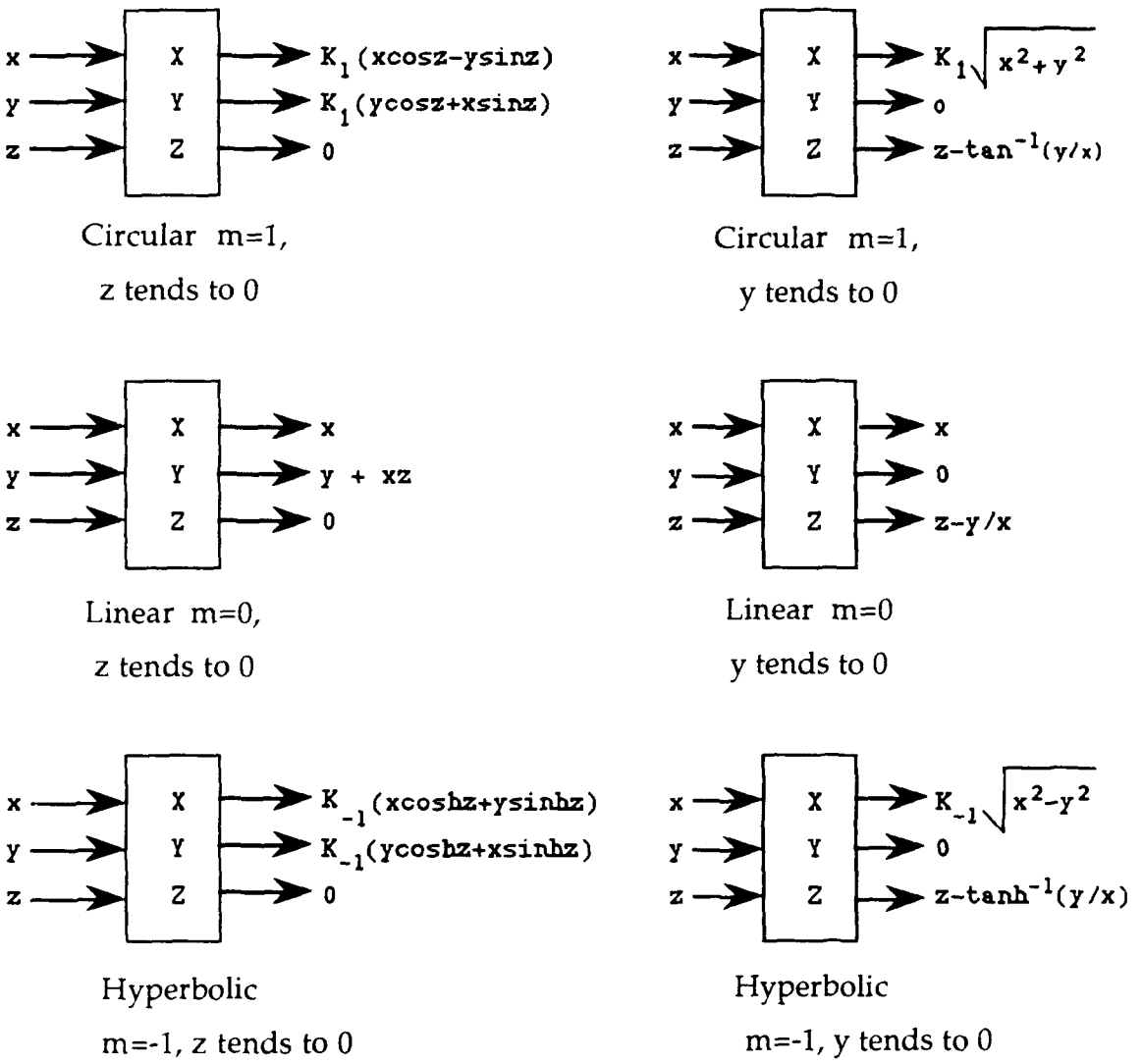


Figure 4.2: Input output functions for CORDIC modes [18]

a speed penalty. It has also been suggested by Ahmed, Ang and Morf [20] to combine a scaling cycle with the CORDIC iteration preceding the current iteration. This procedure overcomes the speed penalty, but at the expense of chip area. Lange et al [21] suggested the implementation of the normalization

factor for K together with the floating point postprocessing at the output. This normalization factor introduces an acceptable relative error of 2^{-16}.

4.3: CORDIC FUNCTIONS AND ACCURACY:

The value 'n' in the above expressions gives the number of CORDIC iterations per operation. This is set by the word length or the number of bits in the inputs x and y, which directly gives the number of iterations. The number of bits is also an index of the CORDIC operation, and accordingly, the number of bits may be chosen for a required accuracy. However, there is a upper limit constraint on having more number of bits and thus more number of iterations since it results in slower operation and consumes more chip area. Hence, there is a trade-off in the selection of word length. The rotations are performed iteratively in many small steps rather than in one single step or one rotation because the sequence of angles $\{\alpha_i\}$ are so chosen that

$$\delta_i = \tan(\alpha_i \sqrt{m}) / \sqrt{m} = 2^{-(i-2)}$$

is an integral power of the machine radix (which is 2 in our case since we are using binary number representation). The multiplication by δ_i in Equations (4.3) and (4.4) may therefore be implemented as a shift operation, one shift per iteration. Note that, for every iteration, we get one bit of the final answer. Hence, we neither require a multiplication facility nor evaluate sines and cosines of arbitrary intermediate angles. Figure 4.2. illustrates two special cases:

1) y is forced to zero : $y_n = 0$

2) z is forced to zero : $z_n = 0$

These two cases generate many elementary functions. Whether y_n or z_n is forced to zero can be taken care of by the proper choice of σ or the direction of rotation.

The identities given in the foot note are used to simplify the above results. It can be seen that by proper choice of the initial values the following functions can be obtained or generated [18].

$x^*z,\ y/x,$

$\sin z,\ \cos z,$

$\sinh z,\ \cosh z,$

$\tan^{-1}(y/x),\ \tanh^{-1}(y/x)$.

$\tan z = \sin z / \cos z$

$\tanh z = \sinh z / \cosh z$

$\exp.z = \sinh z = \cosh z$

$\log_n w = 2\tanh^{-1}[y/x]$

where $x = w+1$ and $y = w-1$

$$\sqrt{w} = \sqrt{x^2 - y^2}$$

where $x = w + 1/4$ and $y = w - 1/4$

Foot note: Mathematical identities:

Let $i = \sqrt{-1}$

$$z = \lim_{m \to 0} i/\sqrt{m} \sin(z\sqrt{m}) \qquad I1$$

$$z = \lim_{m \to 0} 1/\sqrt{m} \tan^{-1} (z\sqrt{m}) \qquad I2$$

$$\sinh z = -i \sin(i z) \qquad I3$$

$$\cosh z = \cos(i z) \qquad I4$$

$$\tanh z = -i \tan^{-1} (i z) \qquad I5$$

4.4: THE CORDIC ITERATION:

In a CORDIC iteration, the new value of the components x and y of any vector P is obtained by adding or subtracting a shifted value of one component to the other. The components are shifted by (i-2) bits to the right, where i is the iteration number, for every iteration and then cross addition or subtraction is performed. These iterations are called microrotations, and every microrotation produces one 'bit' of the answer. Rewriting Equations (4.3) and (4.4), we have,

$$x_{i+1} = x_i - m\,\sigma_i\,2^{-(i-2)}\,y_i \qquad (4.13)$$

$$y_{i+1} = y_i + \sigma_i\,2^{-(i-2)}\,x_i \qquad (4.14)$$

It can be seen that y_{i+1} is obtained by adding a shifted value of x_i to y_i, and similarly, x_{i+1} is obtained by adding a shifted value of y_i to x_i.

By restricting the angular increment α to

$$\alpha_i = \tan^{-1} 2^{-(i-2)}$$

or in general,

$$\alpha_i = m^{-1/2} \tan^{-1} (m^{-1/2}\,2^{-(i-2)})$$

the right hand side of Equations (4.13) and (4.14) can be obtained. In CORDIC, the angles are represented as binary fractions of a half revolution, with the negative angles represented by 2's complements [22]. The angle for the first microrotation is taken to be 90°. To satisfy $\alpha_1 = 90^{\circ}$, and $\sigma = \pm 1$, the condition is

$$-180 \le \theta < +180,$$

which is taken care of by the representation as shown in Figure 4.3.

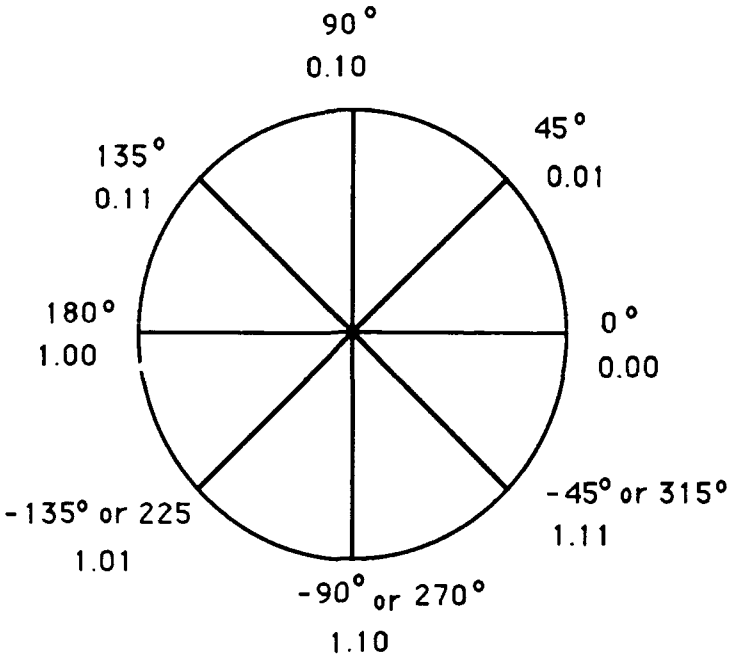


Figure 4.3: Binary representation of angles in CORDIC [22]

4.5: THE ANGLE AND ROTATION COMPUTING SEQUENCE

The CORDIC operation is explained by an example and is illustrated in Tables 4.1 and 4.2. Table 4.1 shows how the angle is computed and Table 4.2 shows how the rotation is performed. The first rows of the Tables show the initial values of the X,Y and Z registers. The X and Y registers have the matrix elements to be rotated as the initial values. In this example, the word length of the three registers are chosen to be 8 bits with the MSB being the sign bit.

4.5.1: Angle computation:

In Table 4.1, the initial value of Z is set to zero. The sign bit of Y_i is used as the reference to determine the operation sign (whether addition or subtraction) to obtain Y_{i+1}, X_{i+1}& Z_{i+1} where i is the index of the CORDIC iteration or microrotation as shown below.

[1] Sign bit of Y_i is '0' which is 'positive'

$X_{i+1} = X_i +$ shifted value of Y_i (addition)

$Y_{i+1} = Y_i -$ shifted value of X_i (subtraction)

$Z_{i+1} = Z_i + \alpha_i$ (addition)

[2] Sign bit of Y_i is '1' which is 'negative'

$X_{i+1} = X_i -$ shifted value of Y_i (subtraction)

$Y_{i+1} = Y_i +$ shifted value of X_i (addition)

$Z_{i+1} = Z_i - \alpha_i$ (subtraction)

For example,the sign bit of Y_3 is 0 indicating positive. The operation sign for Z and X is '+', that is the value of Z_4 is obtained by 'adding' the value of a_3 to Z_3 and the value of X_4 is obtained by adding the value of X_3 to the shifted value of Y_3. The operation sign for Y is the negative of the sign bit and thus, Y_4 is obtained by subtracting the shifted value of X_3 from Y_3 facilitating Y to converge to zero.

As mentioned earlier, the angle α_1 is chosen to be 90^o After the first microrotation, the new value of X is the previous value of Y and the new value

Y is the negative of the previous value of X which conforms to Equation (4.10). The second microrotation is performed by an angle $\tan^{-1}(2^{-(i-2)})$ which is $\tan^{-1}(1)$ since i=2 and thus there are no shifts but only cross addition or subtraction. The number of shifts is given by the exponents of 2 which is the argument of the arctan as seen in the second column of the Table. Since the angle steps are given by the arctan whose arguments are controlled by the iteration number, for any given microrotation, the angle is fixed irrespective of the input values. Thus the angle steps need not be calculated every time but instead, the binary CORDIC representation as shown in Figure 4.3 of the angle steps are stored in the form of look-up tables and recalled at the beginning of every iteration. Since seven bits are used to represent X and Y, seven CORDIC iterations have to be performed for one operation. At the end of seven iterations, the contents of the Z register gives the original angular argument of the co-ordinate components $Y_i$and X_i. In our application, only the value of the angle is necessary and thus the X and Y outputs are ignored. This computed angle is fed as the Z input as shown in Figure 4.4 for the rotation computation described below.

TABLE 4.1

Sequence of steps for computing angle

Angle step	Shift terms	Z-Register	Y-Register	X-Register	Index # i
		0.0000000	0.0101110	1.1000101	1
90	$\tan^{-1}\ \infty$	+0.1000000	-1.1000101	+0.0101110	
		0.1000000	0.0111011	0.0101110	2
45	$\tan^{-1}\ 1$	+0.0100000	-0.0101110	+0.0111011	
		0.1100000	0.0001101	0.1101001	3
26.56	$\tan^{-1}\ 2^{-1}$	+0.0010010	-0.0110100	+0.0000110	
		0.1110010	1.1011001	0.1101111	4
14.04	$\tan^{-1}\ 2^{-2}$	-0.0001001	+0.0011011	-1.1110110	
		0.1101001	1.1110100	0.1111001	5
7.125	$\tan^{-1}\ 2^{-3}$	-0.0000101	+0.0001111	-1.1111110	
		0.1100100	0.0000011	0.1111011	6
3.576	$\tan^{-1}\ 2^{-4}$	+0.0000010	-0.0000111	+0.0000000	
		0.1100110	1.1111100	0.1111011	7
1.7899	$\tan^{-1}\ 2^{-5}$	-0.0000001	+0.0000011	-1.1111111	
148.18		0.1100101	1.1111111	0.1111100	8
$= \theta$		$= \theta$		$= R$	

The index # of the x, y and z register values are indicated in the last column of the table.

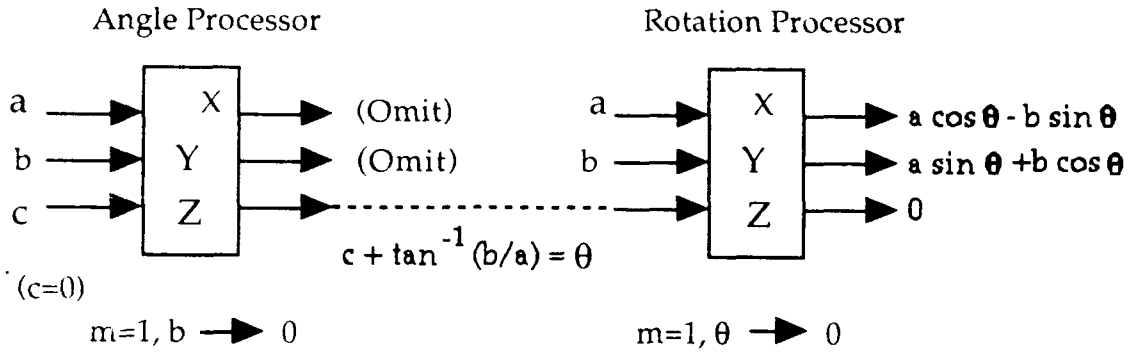


Figure 4.4: Calculation of angle θ, by which the two vectors 'a' and 'b' are rotated.

4.5.2: Rotation computation:

The sequence of steps are shown in Table 4.2. Here, the sign bit of Z_k is used as the reference to determine the operation sign as shown below.

[1] Sign bit of Z_i is '0' which is 'positive'

$X_{i+1} = X_i$ - shifted value of Y_i (subtraction)

$Y_{i+1} = Y_i$ + shifted value of X_i (addition)

$Z_{i+1} = Z_i - \alpha_i$ (subtraction)

[2] Sign bit of Z_i is '1' which is 'negative'

$X_{i+1} = X_i$ + shifted value of Y_i (addition)

$Y_{i+1} = Y_i$ - shifted value of X_i (subtraction)

$Z_{i+1} = Z_i + \alpha_i$ (addition)

The operation sign for Y is the sign indicated by the sign bit of Z and the operation sign of X and Z is the negative of the sign bit of Z. After 7 iterations,

the value of Z is zero, meaning that the rotation has been performed by the complete angle θ until $\theta = \theta \pm \alpha = 0$. Similar steps of the angles α's are used as in the angle module. The contents of the X and Y registers after 7 microrotations are the rotated values of the inputs to the X and Y, by the angle given by the Z register. The Z output is ignored.

TABLE 4.2

Sequence of steps for computing rotation

Angle step	Shift terms	Z-Register	Y-Register	X-register	Index # i
$\theta = 142.18$		0.1100101	0.0101110	1.1000101	1
90°	$\tan^{-1}(\infty)$	-0.1000000	+1.1000101	-0.0101110	
		0.0100101	1.1000101	1.1010010	2
45°	$\tan^{-1}(1)$	-0.0100000	+1.1010010	-1.1000101	
		0.0000101	1.0010111	0.0001101	3
26.56°	$\tan^{-1}(2^{-1})$	-0.0010010	+0.0000110	-1.1001011	
		1.1110011	1.0011101	0.1000010	4
4.04°	$\tan^{-1}(2^{-2})$	+0.0001001	-0.0010000	+1.1100111	
		1.1111100	1.0001101	0.0101001	5
7.125°	$\tan^{-1}(2^{-3})$	+0.0000101	-0.0000101	+1.1110001	
		0.0000001	1.0001000	0.0011010	6
3.576°	$\tan^{-1}(2^{-4})$	-0.0000010	+0.0000001	-1.1111000	
		1.1111111	1.0001001	0.0100010	7
1.7899°	$\tan^{-1}(2^{-5})$	+0.0000001	-0.0000001	+1.1111100	
0°		0.0000000	1.0001000	0.0011110	8
$=\theta$		$=\theta$			

The index # 'i' of the x, y and z register values are indicated in the last column of the table

Thus an operation takes 'i' iterations, i being the word length and it can be seen that in each iteration only shifting and addition / subtraction are performed.

4.6: THE QR ALGORITHM

The QR algorithm has already been described in chapter II. This requires the computation of the following equation, rewritten for quick reference.

$$T_{k+1} = Q_{n-1}^{T} Q_{n-2}^{T} \cdots Q_{1}^{T} T_k Q_1 \cdots Q_{n-2} Q_n$$

The algorithm is explained using an example of a 5 X 5 symmetric, tridiagonal matrix T_1

4.6.1: Computation of eigenvalues:

Eigenvalues are computed in accordance with the above equation and k iterations will be performed. One such iteration is shown in the following example.

$$T_1 = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix} \qquad Q_1^T = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\theta_1 = \tan^{-1}\frac{1}{2} = 26.565^{\circ}$$

θ is calculated based on the Given's rotation. The arguments for the arctan are obtained from the first column of the window selected for the Q^T operation.

$$Q_1^T = \begin{bmatrix} 0.8944 & 0.4472 & 0 & 0 & 0 \\ -0.4472 & 0.8944 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad Q_1^T T_1 = \begin{bmatrix} 2.236 & 1.7888 & 0.4472 & 0 & 0 \\ 0 & 1.34166 & 0.8944 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

$$\theta_2 = \tan^{-1}\frac{1}{1.34166} = 36.6988^o$$

$$Q_2^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.8018 & 0.5976 & 0 & 0 \\ 0 & -0.5976 & 0.8018 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad Q_2^T Q_1^T T_1 = \begin{bmatrix} 2.236 & 1.7888 & 0.4472 & 0 & 0 \\ 0 & 1.6733 & 1.9123 & 0.5976 & 0 \\ 0 & 0 & 1.0691 & 0.8018 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

Observe that the matrix T_1 is being transformed step by step to a upper triangular matrix through row operations by the orthogonal matrices Q^T.

$$\theta_3 = \tan^{-1}\frac{1}{1.0691} = 43.088$$

Now the first two columns are free and we can perform the column operation on the new T_1 by the matrix Q.

$$Q_1=\begin{bmatrix} 0.8944 & -0.4472 & 0 & 0 & 0 \\ 0.4472 & 0.8944 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q_2^T Q_1^T T_1 Q_1=\begin{bmatrix} 2.8 & 0.6 & 0.4472 & 0 & 0 \\ 0.7483 & 1.5 & 1.9124 & 0.5976 & 0 \\ 0 & 0 & 1.0691 & 0.8018 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

$$Q_3^T=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.7303 & 0.6831 & 0 \\ 0 & 0 & -0.6831 & 0.7303 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q_3^T Q_2^T Q_1^T T_1 Q_1=\begin{bmatrix} 2.8 & 0.6 & 0.4472 & 0 & 0 \\ 0.7483 & 1.5 & 1.9124 & 0.5976 & 0 \\ 0 & 0 & 1.4639 & 1.9518 & 0.6831 \\ 0 & 0 & 0 & 0.9135 & 0.7303 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

$$\theta_4 = \tan^{-1}\frac{1}{0.9135} = 47.588^{o}$$

$$Q_3^T Q_2^T Q_1^T T_1 Q_1 Q_2=\begin{bmatrix} 2.8 & 0.7483 & 0 & 0 & 0 \\ 0.7483 & 2.345 & 0.637 & 0.5976 & 0 \\ 0 & 0.8748 & 1.1737 & 1.9518 & 0.68312 \\ 0 & 0 & 0 & 0.9135 & 0.73031 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

$$Q_4^T=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.6744 & 0.7383 \\ 0 & 0 & 0 & -0.73832 & 0.67445 \end{bmatrix}$$

$$Q_4^T Q_3^T Q_2^T Q_1^T T_1 Q_1 Q_2=\begin{bmatrix} 2.8 & 0.7483 & 0 & 0 & 0 \\ 0.7483 & 2.3455 & 0.637 & 0.5976 & 0 \\ 0 & 0.8748 & 1.1737 & 1.9518 & 0.6831 \\ 0 & 0 & 0 & 1.35443 & 1.9692 \\ 0 & 0 & 0 & 0 & 0.8097 \end{bmatrix}$$

$$Q_4^T Q_3^T Q_2^T Q_1^T T_1 Q_1 Q_2 Q_3 = \begin{bmatrix} 2.8 & 0.7483 & 0 & 0 & 0 \\ 0.7483 & 2.3455 & 0.8748 & 0 & 0 \\ 0 & 0.8748 & 2.1905 & 0.6236 & 0.6831 \\ 0 & 0 & 0.9252 & 0.9891 & 1.9692 \\ 0 & 0 & 0 & 0 & 0.8097 \end{bmatrix}$$

$$T_2 = Q_4^T Q_3^T Q_2^T Q_1^T T_1 Q_1 Q_2 Q_3 Q_4 = \begin{bmatrix} 2.8 & 0.7483 & 0 & 0 & 0 \\ 0.7483 & 2.3455 & 0.874 & 0 & 0 \\ 0 & 0.8748 & 2.1905 & 0.925 & 0 \\ 0 & 0 & 0.925 & 2.121 & 0.598 \\ 0 & 0 & 0 & 0.598 & 0.5461 \end{bmatrix}$$

The above matrix is the result of one QR iteration. Observe that it is symmetric and tridiagonal. T_{k+1} will always be symmetrical and tridiagonal as long as the original matrix T_1 is symmetric and tridiagonal. When more iterations are carried out, the off-diagonal entries converge to zero resulting in a diagonal matrix. The diagonal elements will be the eigenvalues of the original matrix as shown below.

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 \\ 0 & 0 & 0 & \lambda_4 & 0 \\ 0 & 0 & 0 & 0 & \lambda_5 \end{bmatrix}$$

4.6.2: Computation of eigenvectors:

The product of all the Q^T matrices which were used for the eigenvalue computation, gives the eigenvectors of the matrix T. Instead of performing matrix multiplications which is both time and area consuming, all the Q^T operations are performed starting with an identity matrix, thus obtaining the same results. One iteration of these computations are shown as an example below.

$$Q_1^T I = \begin{bmatrix} 0.8944 & 0.4472 & 0 & 0 & 0 \\ -0.4472 & 0.8944 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q_2^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.8018 & 0.5796 & 0 & 0 \\ 0 & -0.5796 & 0.8018 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q_2^T Q_1^T I = \begin{bmatrix} 0.8944 & 0.4472 & 0 & 0 & 0 \\ -0.3585 & 0.7171 & 0.5976 & 0 & 0 \\ 0.2672 & -0.5345 & 0.8018 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q_3^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.7303 & 0.6831 & 0 \\ 0 & 0 & -0.6831 & 0.7303 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q_3^T Q_2^T Q_1^T I = \begin{bmatrix} 0.8944 & 0.4472 & 0 & 0 & 0 \\ -0.3585 & 0.7171 & 0.5796 & 0 & 0 \\ 0.1952 & -0.3903 & 0.5856 & 0.6831 & 0 \\ -0.1826 & 0.3651 & 0.5477 & 0.7303 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q_4^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.6744 & 0.7383 \\ 0 & 0 & 0 & -0.7383 & 0.6744 \end{bmatrix}$$

$$X = Q_4^T Q_3^T Q_2^T Q_1^T I = \begin{bmatrix} 0.8944 & 0.4472 & 0 & 0 & 0 \\ -0.3585 & 0.7171 & 0.5976 & 0 & 0 \\ 0.1952 & 0.3903 & 0.5856 & 0.6831 & 0 \\ -0.1231 & 0.2463 & 0.2463 & 0.4926 & 0.7383 \\ 0.1348 & -0.2696 & -0.4044 & -0.5392 & 0.6744 \end{bmatrix}$$

X is the eigenvector matrix with the j^{th} column being the eigenvector corresponding to the eigenvalue λ_j

4.7: STEP BY STEP SUB-ARRAY SELECTION AND PROCESSING

As explained in the previous sections the eigendecomposition by the QR method is performed as a sequence of row operations Q_i^T and column operations Q_i, where i = 1,2,3,....., (m-1) and m is the order of the input matrix. Since we have eight sensors, the input matrix is of the order eight and there are seven row operations and seven column operations. Since the original matrix has been transformed to a tridiagonal form, we have to process at most three elements in every row or column. Each Q_i^T operation is performed on two rows i and (i+1). Every Q_i operation is done on two columns j and (j+1). Thus any Q_i^T or Q_i operation produces at most six new matrix entries and we need to process only these six elements. For the row operation this is achieved by selecting a 2 X 3 window at the top left of the matrix as shown in Figure 4.5. For every consecutive operation, we slide down the window diagonally by one row and one column until the end of the matrix is reached. The last row operation changes only four elements.

Similarly, for the column operations, we select a 3 X 2 window and slide it down diagonally until the end of the matrix is reached. This performs Q_1 to Q_m operations and is shown in Figure 4.6. Note that the Q_1 operation creates only four new elements and thus only two of the three CORDIC processors do the operation and the third processor is fed with zeros as inputs.

For a given i, both Q_i^T and Q_i use the same angles. Thus if all the row operations are carried out first and then followed by the column operations, all the angles in one iteration have to be stored and have to be used in the right order again. To circumvent this drawback, the row and column operations are interleaved. Another advantage of interleaving is that when the angle processor is calculating θ_i for the Q_i^T operation the rotation processors simultaneously perform the Q_{i-2} operation as explained in the next section.

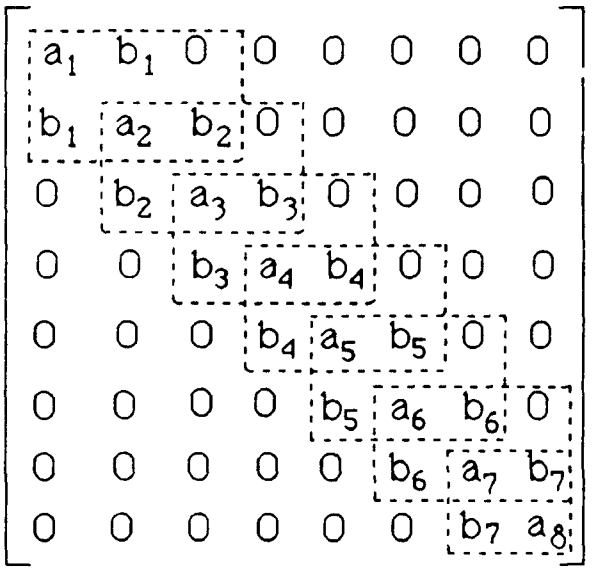


Figure 4.5: Selection of windows for the Q_i^T operation for eigenvalues

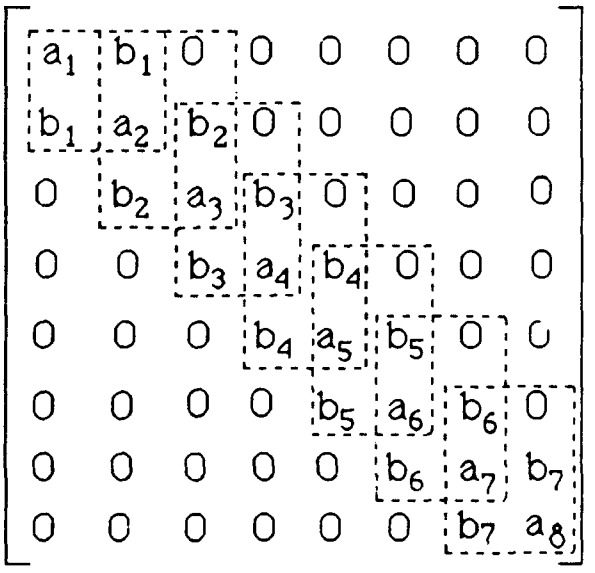


Figure 4.6: Selection of windows for the Q_i operation for eigenvalues

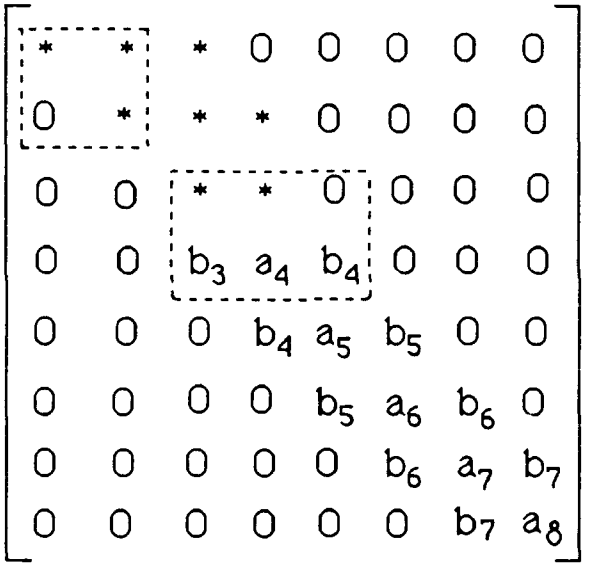


Figure 4.7: First two columns ready for column operation

Note: '*' indicates that those elements have been rotated and their values are different from the original matrix.

After Q_1^T and Q_2^T have been performed, the first two columns of the resulting matrix are free as shown in Figure 4.7 and are available for the Q_1 operation. Meanwhile we calculate the angle for Q_3^T by computing $\tan^{-1}(b_3/a_3)$. This process is continued and when all the Q^T operations are completed, a new iteration is performed starting from row 1 or, a new matrix is loaded row by row if the required number of 'x' iterations are already performed on the previous matrix. Since the loading of the matrices is done row by row, the Q_6 and Q_7 operation on the previous matrix can be performed without being affected by the new iteration or new matrix. After Q_7, we start with Q_1 on the next iteration.

This is explained with the flowchart shown in Figure 4.11.

The eigenvectors are computed by the product of all the Q_i^T. To avoid matrix multiplications, we start with an identity matrix I and perform only the Q_3^T operations on I. To reduce the number of processors for the eigenvector computation by half, the row operations are done in two parts. 2 X 4 windows are selected separately for the left half and the right half of the rows as shown in Figure 4.8. Though we start with only a few non-zero elements in the matrix, we need m/2 = 4 processors because every step and every iteration produces new entries in the resulting matrix.

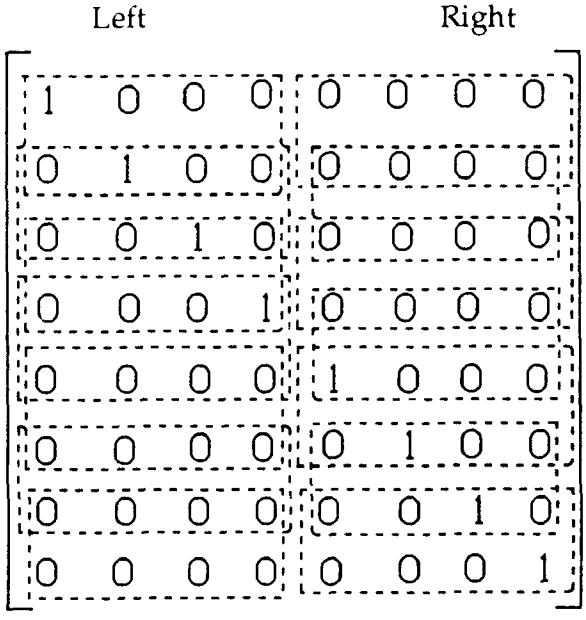


Figure 4.8: Selection of windows for the Q_i^t operation on the identity matrix for the eigenvectors

4.8: DESCRIPTION OF THE PARALLEL HARDWARE BLOCK DIAGRAM

A parallel CORDIC block organization is shown in Figure 4.9. Eigenvalues and eigenvectors are computed in parallel by different sets of processors. The optimized architecture consists of eight processors. The processor P_0 computes the angle θ_i. The processors P_1, P_2 and P_3 compute the Q_i^T and Q_{i-2} operations for the eigenvalues in alternate time steps. Only four processors are required to compute eigenvalues irrespective of the dimension of the matrix as long as m>4.

In the eigenvector module, processors P_4 to P_7 perform the Q_i^T operation starting from the identity matrix. The number of CORDIC processors for eigenvector computation equals m/2. Thus, the total number of processors required to compute both eigenvalues and eigenvectors are $(4 + \frac{m}{2})$.

The input matrix is fed row wise into the memory. The relevant windows are selected with the help of a controller which contains a simple code. The 2 X 3 and 3 X 2 windows are selected alternately and passed to the input registers. Three parallel operations performed by processors P_1 to P_8 are demonstrated by the following Example.

Example:

Assume that Q_1^T, Q_2^T, $Q_3^T$and Q_1 operations have been performed. The next operation that can be performed is Q_2without affecting the row operations and at this instant the second and third columns will be available to perform Q_2 operationas shown in Figure 4.10

[1]To perform the Q_2 operation, we select a 3 X 2 window from the second and third columns. This window is passed to the processors P_1 to P_3. These processors contain the angle θ_2 received from the output of a two stage FIFO

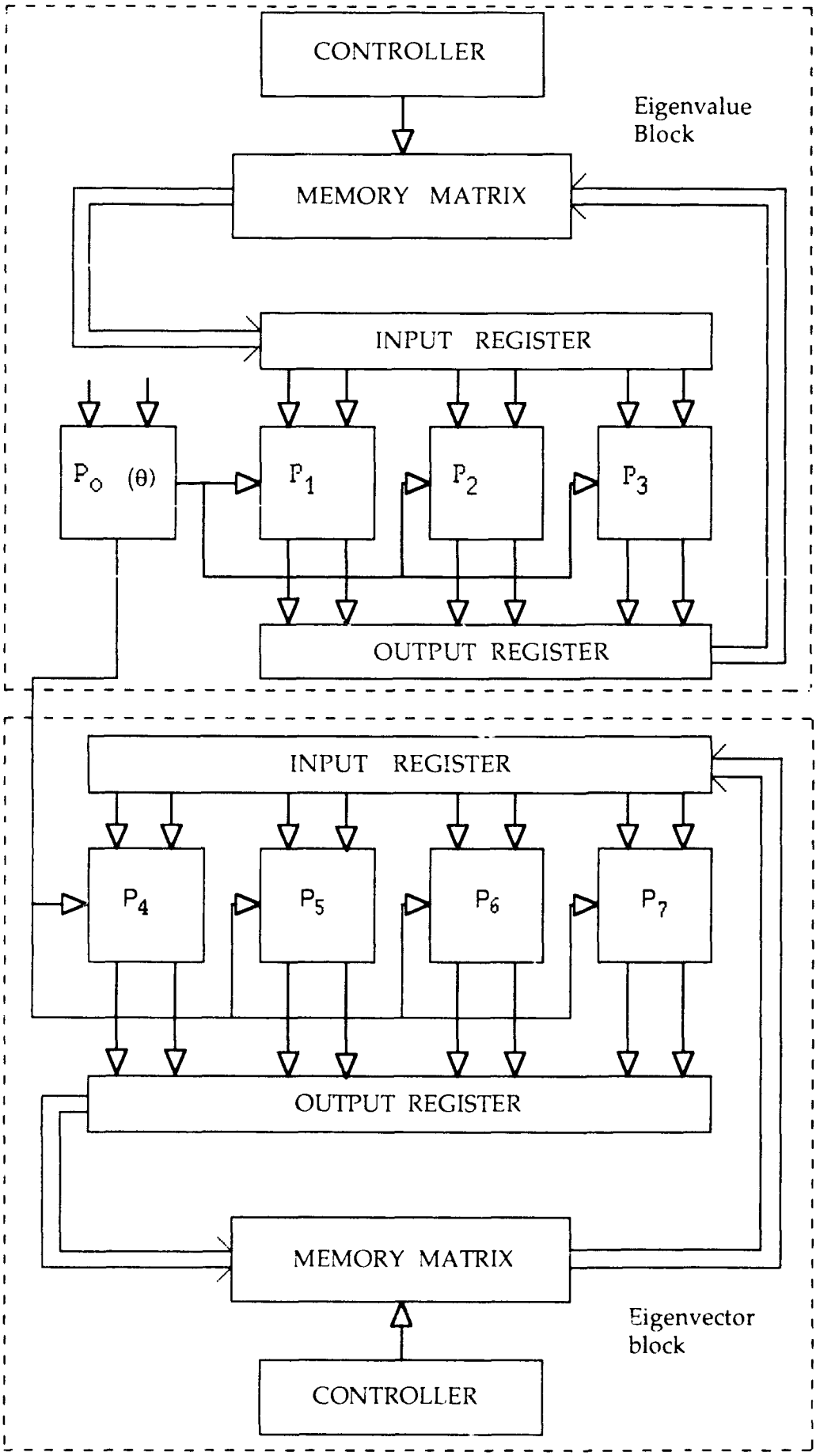


Figure 4.9: Parallel hardware block diagram of a CORDIC block for the computation of QR iterations

connected to P_0.

[2] In the eigenvalue computation mode, the fourth 2 X 3 window is being selected and the first column of this window (which are the elements b_4 and a_4) is sent to P_0 to compute θ_4.

[3]At the same time, P_4 to P_7 contain the third window (right half as shown in Figure 4.7) and contain the angle θ_3 received from the angle processor through a one stage FIFO. It can be seen that the operations that are performed in parallel during this time step are

[1] Angle computation -- θ_4.

[2] Q_2 on columns 2 and 3 in parallel for eigenvalues performed by P_1 to P_3.

[3] Q_3^T on rows 3 and 4 (right half) in parallel for eigenvectors performed by P_4 to P_7

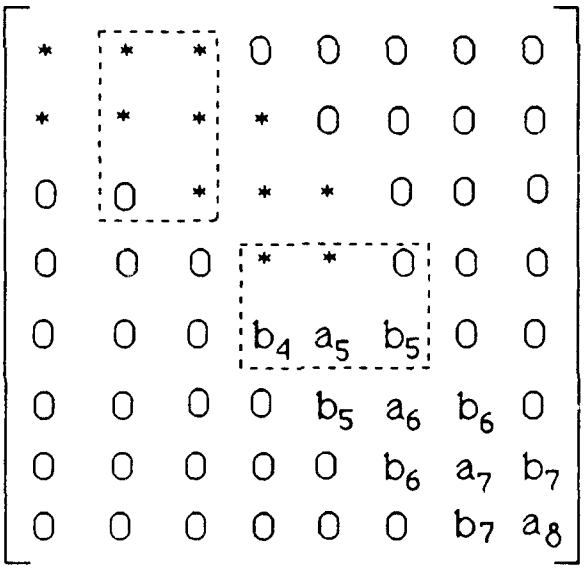


Figure 4.10: Shows the window selected for the row operation Q_4^T while the columns 2 and 3 are free for Q_2 operation.

Note: '*' indicates the elements which have been rotated and updated.

Now, we have $Q_1^T Q_2^T Q_3^T T Q_1 Q_2$ as the new matrix and the angle θ_4 is passed to processors P_1 to P_3 and P_4 to P_7, where Q_4^T will be performed simultaneously. The operations that are performed in parallel during this time step are

[1] Q_4^T on rows 4 and 5 performed by P_1 to P_3.

[2] Q_4^T on rows 4 and 5 (left half) performed by P_4 to P_7.

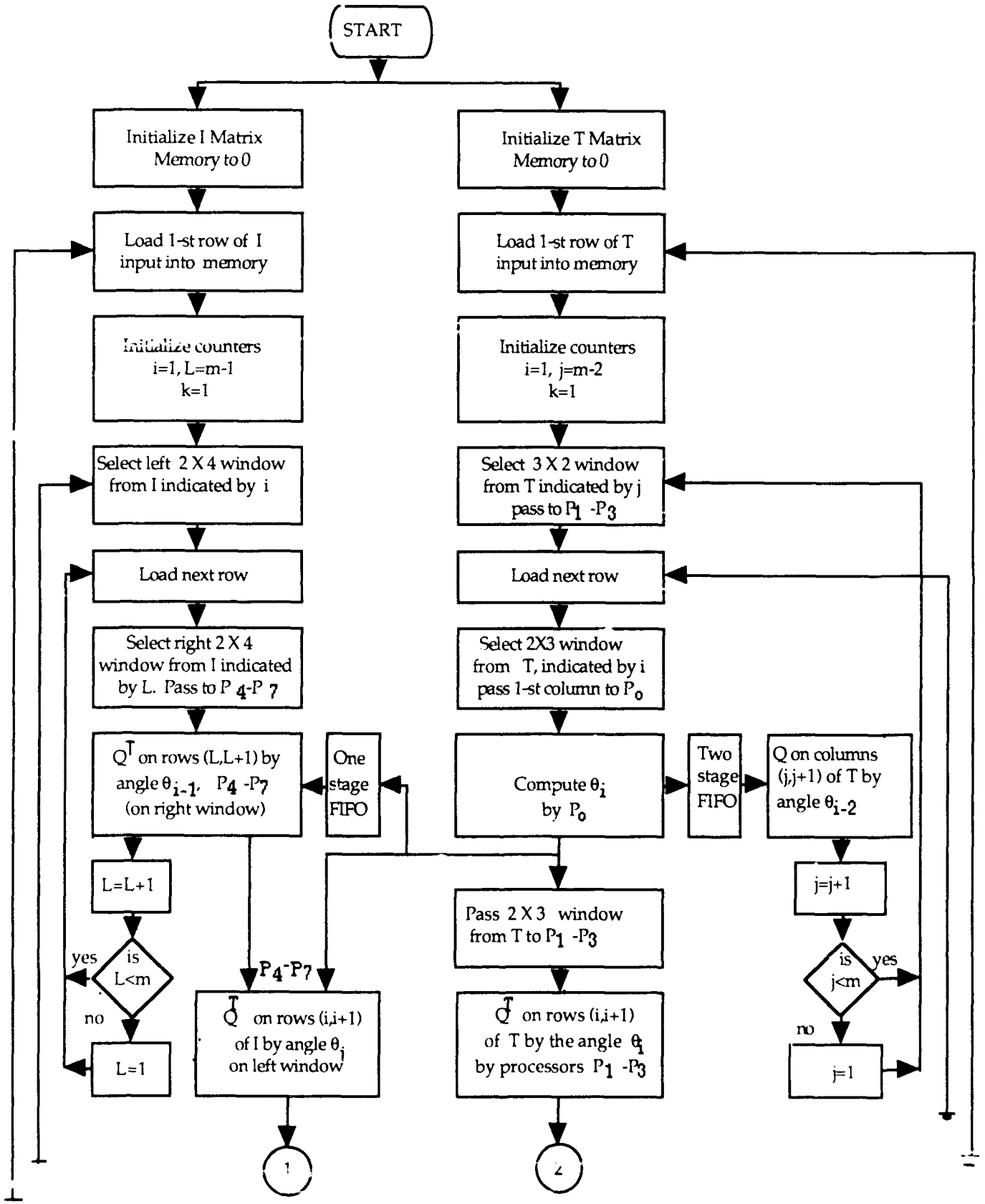
The sequence of operations are explained in the next section.

4.9: PRECEDENCE AND PARALLELISM OF THE COMPUTATIONS

The parallel implementation of the QR decomposition on the CORDIC processors is described with the aid of the flowchart of Figure 4.11. Since eigenvalues and eigenvectors can be computed utilizing the same rotation angle θ, they are shown separately with two parallel paths.

4.9.1: Eigenvalue computation

At the beginning of the computation, the T matrix memory is initialized to 0. Next, the first row of the input T matrix is loaded into the memory. The counters are initialized to i=1, j=m-2 and k=1. Then a 3 X 2 window on columns j and j+1 which are 6 and 7 is selected and passed to the processors P_1 to P_3 for the Q_6 operation. Since, as of now, only one row has been loaded into the memory, the rest of the rows are still zeros and this Q_6 operation does not produce any change in the matrix. After the window has been passed, the next row of T is loaded into the memory. Now we select a 2 X 3 window on T and pass the first column of the window to the processor P_0 for computing θ_1. Next, P_0 will be computing θ_1 while P_1 to P_3 will be computing Q_6. The angle for this Q_6operation is indeterminate since it is just the start of the operation and also


START
Initialize I Matrix Memory to 0
Initialize T Matrix Memory to 0
Load 1-st row of I input into memory
Load 1-st row of T input into memory
Initialize counters i=1, L=m-1 k=1
Initialize counters i=1, j=m-2 k=1
Select left 2 X 4 window from I indicated by i
Select 3 X 2 window from T indicated by j pass to P1 -P3
Load next row
Load next row
Select right 2 X 4 window from I indicated by L. Pass to P 4-P 7
Select 2X3 window from T, indicated by i pass 1-st column to P0
QT on rows (L,L+1) by angle θi-1, P4 -P7 (on right window)
One stage FIFO
Compute θi by P0
Two stage FIFO
Q on columns (j,j+1) of T by angle θi-2
L=L+1
j=j+1
yes
is L<m
no
L=1
P4-P7
Q on rows (i,i+1) of I by angle θj on left window
Pass 2 X 3 window from T to P1 -P3
Q on rows (i,i+1) of T by the angle θi by processors P1 -P3
is j<m
yes
no
j=1
1
2

..... continued

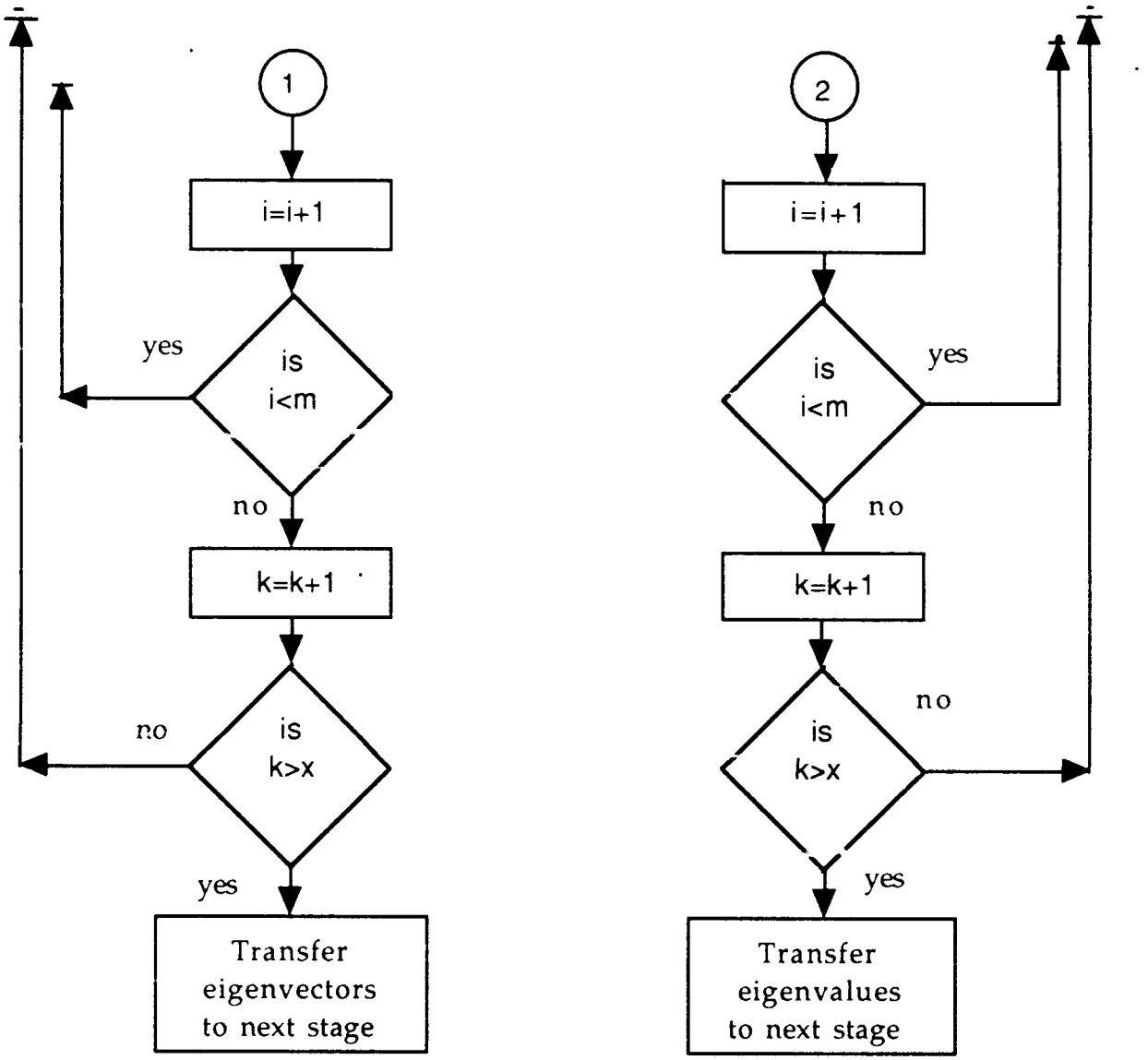


Figure 4.11: Flowchart of operations for computing eigenvalues and eigenvectors

P_1 to P_3 will have zeros as inputs and produce zeros as outputs for any angle. Next, the previously selected 2 X 3 window is passed to the processors P_1 to P_3 and Q_1^T is performed by the angle θ_1 received from P_0. The counters i and j are incremented by one.

Now j=7 which is less than 8 (m=8) the operation is repeated for Q_7. Also i=2 which is less than m, so the above procedure is repeated for Q_2^T. On the next incrementation of i and j, i becomes 3 and j becomes 8 which does not satisfy the condition j<m thus resetting j to 1. This also frees the first two columns and Q_1 operation can be performed. Note that, the Q operation lags behind the Q^T operations by two steps. For example when Q_1^T and Q_2^T are being computed for the 3-rd iteration, Q_6 and Q_7 of the 2-nd iteration are being executed. Q_6 and Q_7 are being computed for the last iteration of one matrix while Q_1^T and Q_2^T are being performed in the first iteration of a new matrix.

When i=m, it indicates that all the rows of the matrix have been covered for the Q^T operation and K is incremented by one. As long as $k \leq x$ (where x is the predetermined number of iterations), the process is continued and when k>x, the eigenvalues are transferred to the next stage. To check the convergence of the decomposition, we have two options.

[1] We can have one CORDIC block and perform unspecified number of iterations until it converges.

[2] We can predetermine the number of iterations x

4.9.2: Eigenvector computation:

This is similar to the eigenvalue computation except that the column operations are not performed. First the memory is initialized to 0. The first row

of the I matrix is loaded and the counters are initialized to i=1, L=m-1 and k=1. The left 2 X 4 window on rows i and i+1 is selected and the next row of the input matrix is loaded into the memory. Next the right 2 X 4 window on rows L and L+1 is selected and passed to the processors P_4 to P_7 and the Q^T operation is performed by the angle θ_{i-1} which is received from the processor P_0 through a one stage FIFO. Then the Q^T operation is performed on the left 2 X 4 window of I by the angle θ_i. The counters L and i are incremented by one and the conditions are checked. The count of L will be 8 in the first pass which is not less than m and L will be reset to 1. When i =m, k is incremented by 1 and once the preset number of iterations are performed, the eigenvectors are transferred to the next stage.

Note that L lags i by one operation meaning that when i=1, Q^T is being done on the left 2 X 4 window of rows i and i+1 and L=7 which does the Q^T on the right 2 X 4 window of rows 7 and 8 of the previous iteration or the last iteration of the previous matrix.

The CORDIC Algorithm is an attractive option considering its simplicity, accuracy and its capability for high speed execution via parallel processing. It is highly flexible and is programmable in the sense that the same processor (or a similar one) can perform a wide variety of functions. In our example, the structure of the processors which do the rotation computation and the angle computation are the same and the only difference is that a few of the input programmable bits are different.

Lange et al [21] have presented a CMOS CORDIC processor which has a throughput of 100ns with 16 bits accuracy and has 72 inputs and 67 outputs.

Considering the degree of advancement of today's CMOS technology, more than 100 CORDIC processors can be fit into one single chip [25]. Depending on the area and speed constraints, we can either select

1] One CORDIC processor block, which performs an unspecified number of iterations until convergence is achieved. In this case, only one matrix can be processed at a time and the next matrix can be loaded only after the current matrix has converged resulting in lower throughput, but consuming less area.

2] We can pre-specify the number of iterations and provide one CORDIC block for every iteration. Here, pipelining is achieved at the cost of area.

Further work on the application of CORDIC approach to MUSIC/ESPIRIT algorithm is in progress.

Chapter V

DOA ESTIMATION FOR WIDEBAND SOURCES

In the previous chapters mainly DOA estimation for narrowband sources are described. Narrowband approaches for DOA estimation have been extended for wideband emitter sources. In the following sections, some of the available wideband DOA algorithms available in the literature are discussed. After studying these algorithms, one of them will be selected for further study to develop real time hardware.

Su and Morf [27] has presented an approach to estimate DOA of wide-band sources. They decompose the rational spectra of the emitters into elementary modes characterized by their poles. The location estimates are derived for each mode using the signal subspace approach. It is claimed in their work that, using modal decomposition signal subspace algorithm asymptotically, exact location estimates can be obtained. In their algorithm more emitters can be resolved than the number of sensors. The emitters may be correlated or coherent and required knowledge of the noise spectra is also minimal. This MDSS algorithm requires the following steps:

1. Estimation of the multichannel covariance sequence.
2. Estimation of poles by overdetermined Yule-Walker method.
3. Estimation of residues.
4. Emitter location estimation from peaks of the spatial spectrum
   i.e. application of MUSIC algorithm.

The previous MDSS algorithm of Su and Morf has been extended for DOA estimation for wide-band signals using the ESPRIT algorithm by Ottersten and Kailath [28]. Their approach is similar to Su and Morf, the only difference is that ESPRIT algorithm is applied instead of MUSIC. Following steps are required for this algorithm

1. Estimate the covariance of the output of the array.
2. Estimate the system poles and residues.
3. Determine the dimension of the signal subspace at the system poles and find vectors that span this space.
4. Estimate the angles of arrival using ESPRIT.

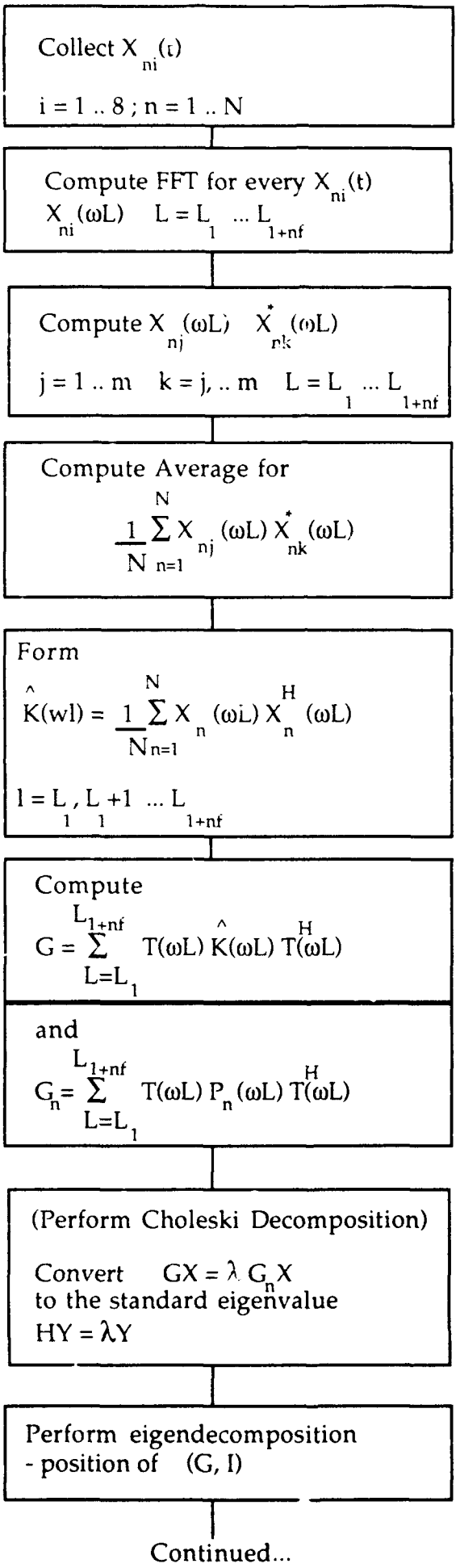
Wang and Kaveh [39] have proposed another method of detection and estimating the DOA of multiple wide-band sources. Their method requires computation of discrete Fourier transform on the output of the array. Covariance is estimated in different frequency bins. The approximate transformation matrices are computed using initial estimates of DOA's and the array manifold. The signal subspaces at discrete frequencies are estimated. The eigendecomposition is performed and then MUSIC algorithm is applied to find DOAs.

An extension of Wang and Kaveh method [40] has been proposed by Shaw and Kumaresan [26]. They used a simple bilinear transformation matrix and the approximation resulting from dense and equally spaced array structure has been used. This is done to combine the individual narrowband spectral matrices for coherent processing. This method is non-iterative and initial estimates of the DOA are not required.

This method of DOA estimation of wideband signals seems very attractive

and was easy to simplify. A pipelined flowchart of this method is shown in Figure 5.1. In this flowchart, it is assumed that there are 8 sensors. First of all data is collected from all the sensors. It is also assumed that data is collected for 64 segments and each segment will consist of 64 samples. FFT is performed and then covariance matrices are computed which are then averaged. G and G_n are computed in parallel each requiring two matrix multiplications. A Choleski decomposition is performed to convert G into standard form which will make eigendecomposition little easier. Eigendecomposition can be performed by previously discussed Householder transformation and QR method. Using eigenvalues, number of sources are estimated and then finally angle of arrival can be estimated.

At this time we are further studying this algorithm in order to develop its architecture.


Collect $X_{ni}(t)$
$i = 1 .. 8 ; n = 1 .. N$
Compute FFT for every $X_{ni}(t)$
$X_{ni}(\omega L) \quad L = L_1 \ldots L_{1+nf}$
Compute $X_{nj}(\omega L) \quad X^*_{nk}(\omega L)$
$j = 1 .. m \quad k = j, .. m \quad L = L_1 \ldots L_{1+nf}$
Compute Average for
$\frac{1}{N} \sum_{n=1}^{N} X_{nj}(\omega L) X^*_{nk}(\omega L)$
Form
$\hat{K}(wl) = \frac{1}{N} \sum_{n=1}^{N} X_n(\omega L) X^H_n(\omega L)$
$l = L_1, L_1 + 1 \ldots L_{1+nf}$
Compute
$G = \sum_{L=L_1}^{L_{1+nf}} T(\omega L) \hat{K}(\omega L) T^H(\omega L)$
and
$G_n = \sum_{L=L_1}^{L_{1+nf}} T(\omega L) P_n(\omega L) T^H(\omega L)$
(Perform Choleski Decomposition)
Convert $GX = \lambda G_n X$
to the standard eigenvalue
$HY = \lambda Y$
Perform eigendecomposition
- position of (G, I)
Continued...

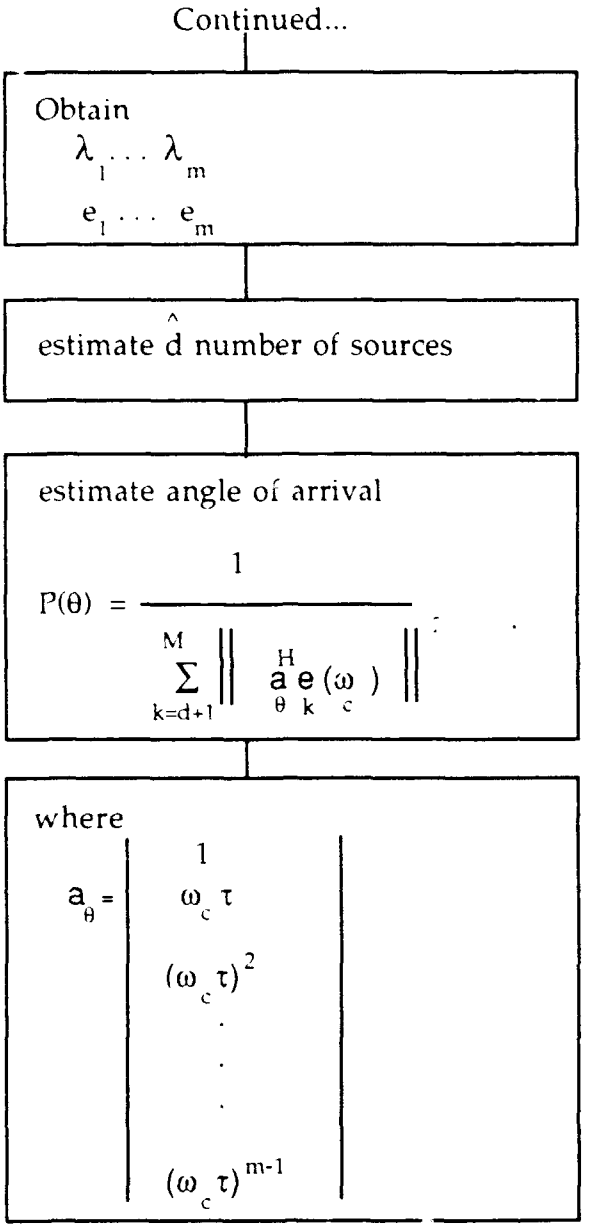


Figure 5.1 Estimation of angles of arrivals of broadband signals

Chapter VI

CONCLUSIONS

The work performed for the development of parallel architecture for sensor array algorithms for the last six months has been described in section 6.1 and future work is given in section 6.2. Further results, recommendations, suggestions and conclusions will be presented in the final report.

6.2:WORK PERFORMED

1. A Literature survey has been performed and investigated for various algorithms available for narrow band and wide band cases.

2. In the area of narrow band, MUSIC and ESPRIT algorithms were selected and further studied. These algorithm were converted into computationaly efficient algorithms and subsequently parallelized. Three different architectures namely Systolic architectures, Cordic processors and SIMD are under consideration.

3.These algorithms required eigenvalue decomposition. Householder transformation was used to convert covariance matrix into tridiagonal matrix. The QR method was used to finally obtain eigenvalues and eigenvectors. The detailed systolic architecture is being developed for parallelized Householder transformations and QR method.

4. Single instruction multiple data (SIMD) type of architectures lend themselves for the implementation of narrow band DOA estimation. The computation of covariance matrix, Householders transformation and QR method con be easily computed using SIMD machine. The work on SIMD machine is under progress.

5. A third approach of utilizing cordic processors is also being investigated for the implementation of eigendecomposition.

6. In the case of wideband DOA estimations, various algorithms available in the literature were studied. It was found that wideband DOA estimation is more computationaly intensive than narrow band case. An algorithm proposed by Shaw has been selected for further study. This algorithm again has been modified and substituted with computionally efficient operations. A parallelized flowchart has been developed. A block diagram for its hardware has also been developed. Other available algorithms are also under investigation.

6.3: FUTURE WORK

1. Detailed architecture design will be done for both MUSIC and ESPRIT algorithms.
2. Interface modules will be designed to transfer data from one computational module to another.
3. High level simulation will be performed for MUSIC and ESPRIT to check finite precision effects and finalize word length for various modules.
4. A simulation will be performed for the architecture itself using VHDL language of Workview 4.0 software from Viewlogic.
5. A detailed architecture will also be developed for the wideband case.
6. An algorithm equivalent to QR method is being investigated which may be more efficient than the presently proposed algorithm.
7. Modifications to ESPRIT algorithm are being studied.
8. A study will be performed to estimate real time requirements for the computations of DOA. Based on this study, real time architecture for the computation of MUSIC and ESPRIT will be proposed.

REFERENCES

[1] C. H. Knapp and G. C. Carter, "The generalized correlation method for estimation of time delay, " IEEE Trans. Acoustic, Speech and Signal Processing, Vol. ASSP-24, pp. 320-327, Aug. 1976.

[2] J. Capon, "High resolution frequency-wavenumber spectrum analysis. Proc. IEEE, Vol. 57, pp. 1408-1418, Aug. 1969.

[3] J. P. Burg, "Maximum entropy spectral analysis," in Proc. 37th Ann. Int. SEG Meet. Oklahoma City, OK, 1967.

[4] R. O. Schmith, "Multiple emitter location and signal parameter estimation," IEEE Trans. on Antennas and Propagation, Vol AP-34 No. 3., pp. 276-280, Mar. 1986.

[5] R. Roy and T. Kailath, "ESPRIT-Estimation of signal parameters via rotational invariance techniques, " in proc. IEEE Trans. Acoustic, Speech and Signal Processing, Vol. 37, No. 7, pp. 984-995, July 1989.

[6] D. Spielman, A. Paulraj, "Performance analysis of the MUSIC algorithm," in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Tokyo, Japan, pp 1909-1912, Apr 1986 .

[7] T. J. Shan, A. Paulraj, "On smoothed rank profile tests in eigenstructure approach to direction-of-arrival estimation," in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Tokyo, Japan, pp 1905-1908, Apr 1986.

[8] G.H. Golub, C. F. Van Loan " An analysis of the total least square problem, " SIAM J. Numerical Analysis. Vol 17. N 17, December 1980.

[9] C. Y. Chen and J. A. Abraham, "Fault -tolerant systems for computations of eigenvalues and singular value" SPIE Vol.696 Advanced algorithm and architecture for signal processing, pp 228-237, 1986

[10] J. J. Dongarra and D. C. Sorensen, "On the implementation of fully parallel *algorithm for symmetric eigenvalue problem", SPIE Vol. 696 Advanced* algorithm and architecture for signal processing, pp 45-53, 1986

[11] J. H. Wilkinson, "The algebriac eigenvalue problem," Clarendon Press, Oxford, Chapter 4, 1965

[12] C. F. T. Tang, K. J. R. Liu and S. A. Tretter, "On systolic array for recursive complex Householder transformations with applications to array processing.", in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Toronto, Canada, pp 1033-1036, Apr 1991

[13] K. J. R. Liu, S. F. Heieh, K. Yao,"Two level pipelined implementation of systolic block Householder transformations."in proc. IEEE Conf. Acoustic, Speech and Signal Processing, pp 1631-1634, Alburquerque, NM. Apr 1990 .

[14] S. Y. Kung VLSI array processors, Prentice Hall, Englewood Cliffs, NJ. 1987.

[15] W.Phillips and W.Robertson,"Systolic architecture for symmetric tridiagonal eigenvalue problem", IEEE International conference on systolic arrays, pp. 145-

150, 1988.

[16] K.J.R.Liu and K.Yao, "Multiphase systolic architecture for spectral decomposition," 1990 International conference on parallel processing, pp I-123 to I-126.

[17] Volder J.E., "The CORDIC trigonometric computing technique", IRE transactions. electronic computers, EC-8, No.3, pp. 330-334, Joint computer conference, San Fransisco, California, 1956.

[18] Walther J.S., "A unified algorithm for elementary functions" Spring joint computer conference, AFIPS conference proceedings, 38, pp. 379-385, 1971.

[19] Gene L. Haviland and Al A. Tuszynski, "A CORDIC arithmetic processor chip", IEEE transactions on computers, Vol.c-29, No.2, pp. 68-79, February 1980.

[20] H.M. Ahmed et al, "A VLSI speech analysis chip set utilizing co-ordinate rotation arithmetic", IEEE Int. Symp. on Circuits and Systems, pp. 737-741, 1981.

[21] A. A. J. de Lange et al, "An optimal floating-point pipeline CMOS CORDIC processor", IEEE Int. Symp. on Circuits and Systems, pp. 2043-2047, 1988.

[22] D. H. Daggett, "Decimal-binary conversions in CORDIC", IRE transactions on electronic computers, EC-8, No.3, pp. 335-339, Joint computer conference, San Fransisco, California, 1956.

[23] Ahmed et al, "Highly concurrent computing structures for matrix arithmetic

and signal processing", IEEE Computer, pp. 65-81, January 1982.

[24] Milos D. Ercegovac and Tomas Lang, "Implementation of fast angle calculation and rotation using on-line CORDIC", IEEE Int. Symp. on Circuits and Systems, pp. 2703-2706, 1988.

[25] S.C. Bass et al, "A bit-serial, floating-point CORDIC processor in VLSI", IEEE Int. Conf. on Acoustics, Speech and Signal Processing, V3.1, PP. 1165-1168, Toronto, Canada, 1991.

[26] Arnab K. Shaw and Ramdas Kumaresan, "Estimation of angles of arrivals of broadband signals", IEEE ICASSP-87, pp.2296-2299, 1987.

[27] Guaning Su and Martin Morf, "modal decomposition signal subspace algorithms", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. ASSP-34, No. 3, pp. 585-602, June 1986.

[28] Bjorn Ottersten and Thomas Kailath, "Direction-of-arrival estimation for wide-band signals using the ESPRIT algorithm", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. 38, No. 2, pp. 317-327, February 1990.

[29] Kevin M. Buckley and LLoyd J. Griffiths, "Broad-band signal-subspace spatial-spectrum (BASE-ALE) estimation", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. 36, No. 7, July 1988.

[30] G. C. Carter, [Editor], "Special issue on time-delay estimation", IEEE Trans. on ASSP, VOL. 29, No. 3, 1981.

[31] C. H. Knapp and G. C. Carter, "The generalized correlation...", IEEE Trans. ASSP, VOL. 24, No. 4, pp. 320-237, 1976.

[32] W. J. Bangs and P. Schultheiss, "Space-Time processing...", in *Signal Processing*, J. W. R. Griffiths et al, Eds. New York, Academic Press, pp. 577-590, 1973.

[33] W. R. Hahn and S. A. Tretter, "Optimum processing for ...", IEEE Trans. IT, VOL. 19, No. 5, pp. 608-614.

[34] M. Wax and T. Kailath, "Optimum localizations of multiple source by passive arrays", in proc. IEEE Trans. Acoustic, Speech and Signal Processing vol. ASSP-31, No. 5, pp. 1210-1218, Oct. 1983.

[35] M. Morf. et al, "Investigation of new algorithms ...", DARPA Tech. Rept., No. M-355-1.

[36] B. Porat and B. Frienlander, "Estimation of spatial and spectral parameters of multiple sources", IEEE Trans. on info. theory, vol. IT-29, pp. 412-425, May 1983.

[37] A. Nehorai, G. Su, M. Morf, "Estimation of time difference of arrivals for multiple ARMA sources by pole decomposition", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-31, pp. 1478-1491, Dec. 1983.

[38] M. Wax T. J. Shan and T.Kailath, "Spatio-temporal spectral analysis by eigenstructure method", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-32, No. 4, Aug. 1984.

[39] H. Wang and M. Kaveh, "Estimation of angles-of -arrival for wide-band sources", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, pp. 7.5.1-7.5.4, Mar. 19-21, 1984.

[40] H. Wang and M. Kaveh, "Coherent Signal Subspace processing for the detection and estimation of angle of arrival of multiple wide-band sources", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-33, No. 4, pp. 823-831, Aug. 1985.

[41] T. L. Henderson, "Rank Reduction for Broadband ...", in proc. *19th Asilomar Conf. on Circ., Syst. & Comp.*, Nov. 1985.